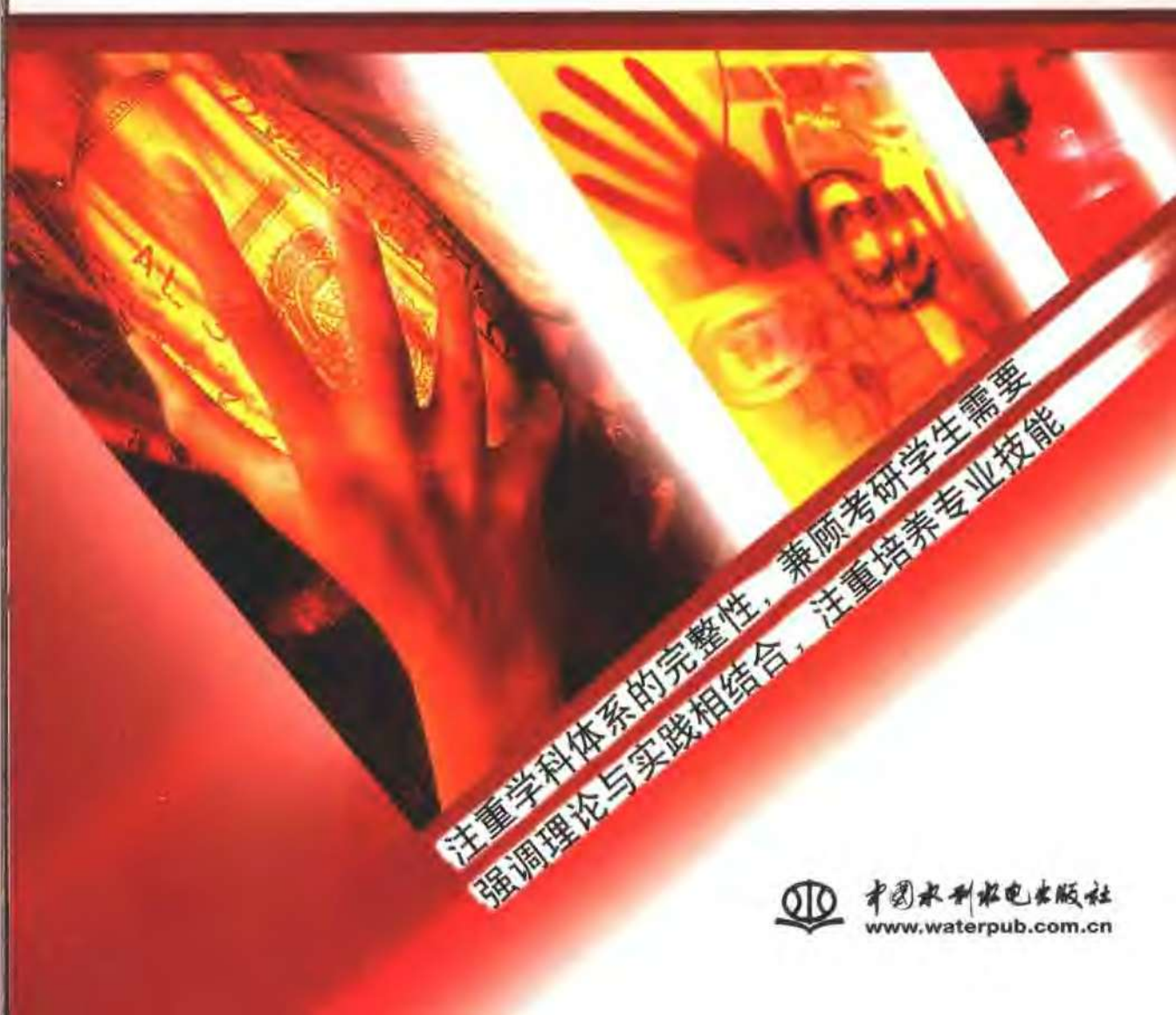
21世纪

高等院校规划教材

王 红 主 编
侯 刚 张凤云 杨德芳 副主编

操作系统原理及应用(Linux)



中国水利水电出版社
www.waterpub.com.cn

责任编辑：宋俊娥　加工编辑：杨元佳　封面设计：俞兆君

- 采用“任务驱动”的编写方式，引入案例和启发式教学方法
- 提供电子教案、案例素材等教学资源，教材立体化配套
- 满足高等院校应用型人才培养的需要

大学计算机基础
办公自动化实用技术
计算机科学导论
计算机专业英语
C语言程序设计
C++程序设计
C#程序设计
SQL Server 2000数据库及应用

Visual C++程序设计及应用
Visual Basic 6.0程序设计及应用
Visual Basic.NET程序设计
Visual FoxPro 6.0程序设计及应用
Java程序设计及应用
ASP程序设计及应用
ASP.NET程序设计及应用
PowerBuilder程序设计及应用

Web程序设计及应用
数据结构（C++语言描述）
数据结构（C语言描述）
微机原理与汇编语言
微机原理及接口技术
汇编语言程序设计
操作系统原理及应用(Linux)
Access 2002数据库及应用
大学信息技术概论

数据库原理及应用（SQL Server 2000）
数据库原理及开发（SQL Server+PowerBuilder）
Web数据库与XML应用
网页设计与制作实用技术
动态网页设计实用技术
移动通信
电子技术
电工技术

网站建设与管理
数据通信与计算机网络
脚本语言与动态网页设计
计算机网络实用技术
局域网技术与组网工程
计算机网络安全技术
综合布线技术与施工
网络互联技术

单片机原理与应用
DSP原理及应用
软件工程
微机组装与维护实用技术
多媒体技术与应用
计算机控制与仿真技术
电子商务概论
管理信息系统

北京万水电子信息有限公司
Beijing Multi-Channel Electronic Information Co.,Ltd.
地址　北京市海淀区长春桥路5号新起点嘉园4号楼1706室
电话　010-82562819(总机)
传真　010-82564371
邮编　100089
E-mail: mchannel@263.net

ISBN 7-5084-2912-5
9 787508 429120 >

ISBN 7-5084-2912-5
定价：26.00 元

21世纪高等院校规划教材

操作系统原理及应用（Linux）

王 红 主 编

侯 刚 张凤云 杨德芳 副主编

中国水利水电出版社

内 容 提 要

本书介绍操作系统的基本原理及应用，并以 Linux 操作系统作为具体实例加以说明。全书共 9 章，主要内容包括操作系统的基本概念、功能、操作系统的发展史及 Linux 操作系统概述，进程的概念、进程描述、进程控制、进程的同步与互斥、进程通信及线程，作业调度及进程调度、死锁的概念与产生原因以及解决死锁的方法，各种内存管理及分配方法的思想、数据结构、重定位及实现原理，文件的逻辑结构、物理结构及文件系统的构成，操作系统对设备的控制、分配、缓冲区的管理等，Linux 系统的网络功能，现代流行的 UNIX、Windows 2000 操作系统的特点、原理以及分布式操作系统的概况，操作系统的安全性。

本书例题及课后习题丰富，便于初学者对操作系统原理进行理解、掌握并运用；在配套教材《操作系统原理及应用——学习指导与题解》中，为各章提供了内容要点复习、教材习题解答、新增习题及解答，并为各部分原理提供了一些实训内容。

本书内容全面，选材适当，结构合理，理论结合实际，可作为应用型本科、成人高校、高职高专院校计算机及相关专业的操作系统课程的教材，也可以作为相关专业的师生及专业技术人员的参考书。

本书所配电子教案可以从中国水利水电出版社网站下载，网址为 http://www.waterpub.com.cn/softdown/。

图书在版编目（CIP）数据

操作系统原理及应用：Linux / 王红主编. —北京：中国水利水电出版社，2005

（21 世纪高等院校规划教材）

ISBN 7-5084-2912-5

Ⅰ.操… Ⅱ.王… Ⅲ.Linux 操作系统—高等学校—教材 Ⅳ.TP316.89

中国版本图书馆 CIP 数据核字（2005）第 057100 号

书　　名	**操作系统原理及应用（Linux）**
作　　者	王　红　主编　侯　刚　张凤云　杨德芳　副主编
出版　发行	中国水利水电出版社（北京市三里河路 6 号　100044）
	网址：www.waterpub.com.cn
	E-mail：mchannel@263.net（万水）
	sales@waterpub.com.cn
	电话：（010）63202266（总机）、68331835（营销中心）、82562819（万水）
经　　售	全国各地新华书店和相关出版物销售网点
排　　版	北京万水电子信息有限公司
印　　刷	北京市天竺颖华印刷厂
规　　格	787mm×1092mm　16 开本　18.25 印张　404 千字
版　　次	2005 年 6 月第 1 版　2005 年 6 月第 1 次印刷
印　　数	0001—5000 册
定　　价	26.00 元

凡购买我社图书，如有缺页、倒页、脱页的，本社营销中心负责调换

版权所有 · 侵权必究

序

随着计算机科学与技术的飞速发展，计算机的应用已经渗透到国民经济与人们生活的各个角落，正在日益改变着传统的人类工作方式和生活方式。在我国高等教育逐步实现大众化后，越来越多的高等院校会面向国民经济发展的第一线，为行业、企业培养各级各类高级应用型专门人才。为了大力推广计算机应用技术，更好地适应当前我国高等教育的跨跃式发展，满足我国高等院校从精英教育向大众化教育的转变，符合社会对高等院校应用型人才培养的各类要求，我们成立了“21 世纪高等院校规划教材编委会”，在明确了高等院校应用型人才培养模式、培养目标、教学内容和课程体系的框架下，组织编写了本套“21 世纪高等院校规划教材”。

众所周知，教材建设作为保证和提高教学质量的重要支柱及基础，作为体现教学内容和教学方法的知识载体，在当前培养应用型人才中的作用是显而易见的。探索和建设适应新世纪我国高等院校应用型人才培养体系需要的配套教材已经成为当前我国高等院校教学改革和教材建设工作面临的紧迫任务。因此，编委会经过大量的前期调研和策划，在广泛了解各高等院校的教学现状、市场需求，探讨课程设置、研究课程体系的基础上，组织一批具备较高的学术水平、丰富的教学经验、较强的工程实践能力的学术带头人、科研人员和主要从事该课程教学的骨干教师编写出一批有特色、适用性强的计算机类公共基础课、技术基础课、专业及应用技术课的教材以及相应的教学辅导书，以满足目前高等院校应用型人才培养的需要。本套教材消化和吸收了多年来已有的应用型人才培养的探索与实践成果，紧密结合经济全球化时代高等院校应用型人才培养工作的实际需要，努力实践，大胆创新。教材编写采用整体规划、分步实施、滚动立项的方式，分期分批地启动编写计划，编写大纲的确定以及教材风格的定位均经过编委会多次认真讨论，以确保该套教材的高质量和实用性。

教材编委会分析研究了应用型人才与研究型人才在培养目标、课程体系和内容编排上的区别，分别提出了 3 个层面上的要求：在专业基础类课程层面上，既要保持学科体系的完整性，使学生打下较为扎实的专业基础，为后续课程的学习做好铺垫，更要突出应用特色，理论联系实际，并与工程实践相结合，适当压缩过多过深的公式推导与原理性分析，兼顾考研学生的需要，以原理和公式结论的应用为突破口，注重它们的应用环境和方法；在程序设计类课程层面上，把握程序设计方法和思路，注重程序设计实践训练，引入典型的程序设计案例，将程序设计类课程的学习融入案例的研究和解决过程中，以学生实际编程解决问题的能力为突破口，注重程序设计算法的实现；在专业技术应用层面上，积极引入工程案例，以培养学生解决工程实际问题的能力为突破口，加大实践教学内容的比重，增加新技术、新知识、新工艺的内容。

本套规划教材的编写原则是：

在编写中重视基础，循序渐进，内容精炼，重点突出，融入学科方法论内容和科学理念，反映计算机技术发展要求，倡导理论联系实际和科学的思想方法，体现一级学科知识组织的层次结构。主要表现在：背靠计算机学科的科学体系，明确目标定位，分类组织实施，兼容互补；理论与实践并重，强调理论与实践相结合，突出学科发展特点，体现学科

发展的内在规律：教材内容循序渐进，保证学术深度，减少知识重复，前后相互呼应，内容编排合理，整体结构完整；采取自顶向下设计方法，内涵发展优先，突出学科方法论，强调知识体系可扩展的原则。

本套规划教材的主要特点是：

（1）面向应用型高等院校，在保证学科体系完整的基础上不过度强调理论的深度和难度，注重应用型人才的专业技能和工程实用技术的培养。在课程体系方面打破传统的研究型人才培养体系，根据社会经济发展对行业、企业的工程技术需要，建立新的课程体系，并在教材中反映出来。

（2）教材的理论知识包括了高等院校学生必须具备的科学、工程、技术等方面的要求，知识点不要求大而全，但一定要讲透，使学生真正掌握。同时注重理论知识与实践相结合，使学生通过实践深化对理论的理解，学会并掌握理论方法的实际运用。

（3）在教材中加大能力训练部分的比重，使学生比较熟练地应用计算机知识和技术解决实际问题，既注重培养学生分析问题的能力，也注重培养学生思考问题、解决问题的能力。

（4）教材采用“任务驱动”的编写方式，以实际问题引出相关原理和概念，在讲述实例的过程中将本章的知识点融入，通过分析归纳，介绍解决工程实际问题的思想和方法，然后进行概括总结，使教材内容层次清晰，脉络分明，可读性、可操作性强。同时，引入案例教学和启发式教学方法，便于激发学习兴趣。

（5）教材在内容编排上，力求由浅入深，循序渐进，举一反三，突出重点，通俗易懂。采用模块化结构，兼顾不同层次的需求，在具体授课时可根据各校的教学计划在内容上适当加以取舍。此外还注重了配套教材的编写，如课程学习辅导、实验指导、综合实训、课程设计指导等，注重多媒体的教学方式以及配套课件的制作。

（6）大部分教材配有电子教案，以使教材向多元化、多媒体化发展，满足广大教师进行多媒体教学的需要。电子教案用 PowerPoint 制作，教师可根据授课情况任意修改。相关教案的具体情况请到中国水利水电出版社网站 www.waterpub.com.cn 下载。此外还提供相关教材中所有程序的源代码，方便教师直接切换到系统环境中教学，提高教学效果。

总之，本套规划教材凝聚了众多长期在教学、科研一线工作的教师及科研人员的教学科研经验和智慧，内容新颖，结构完整，概念清晰，深入浅出，通俗易懂，可读性、可操作性和实用性强。本套规划教材适用于应用型高等院校各专业，也可作为本科院校举办的应用技术专业的课程教材，此外还可作为职业技术学院和民办高校、成人教育的教材以及从事工程应用的技术人员的自学参考资料。

我们感谢该套规划教材的各位作者为教材的出版所做出的贡献，也感谢中国水利水电出版社为选题、立项、编审所做出的努力。我们相信，随着我国高等教育的不断发展和高校教学改革的不断深入，具有示范性并适应应用型人才培养的精品课程教材必将进一步促进我国高等院校教学质量的提高。

我们期待广大读者对本套规划教材提出宝贵意见，以便进一步修订，使该套规划教材不断完善。

21 世纪高等院校规划教材编委会

2004 年 8 月

前　言

计算机操作系统管理计算机系统所有的软件和硬件资源，同时为用户提供一个方便、安全、可靠的工作环境。操作系统课程是计算机专业学生的必修课程，掌握并理解计算机操作系统的基本原理和方法，对计算机专业的学生和技术人员来说是非常必要的。

由于操作系统在计算机系统中所处的特殊地位，以及它具有的抽象性及理论深度，使得这门课程并不是那么容易真正学懂弄通。很多初学者会感到这门课程理论性太强、概念原理太多，不容易记住，更不易掌握，特别是在本科操作系统课程中，选择合适的实验内容是一个普遍的难题。编者正是在这种情况下，在结合多年操作系统课程教学经验的基础上对该教材进行了编写。

本书主要讲述操作系统的基本原理及应用，具体实例采用 Linux 操作系统。内容共分 9 章，第 1 章介绍操作系统的基本概念、功能、操作系统的发展史及 Linux 操作系统概述；第 2 章介绍进程的概念、进程描述、进程控制、进程的同步与互斥、进程通信及线程；第 3 章介绍作业调度及进程调度，死锁的概念与产生的原因，以及解决死锁的方法；第 4 章介绍操作系统对内存的管理方法，主要介绍各种内存管理及分配方法的思想、数据结构、重定位及实现原理；第 5 章介绍操作系统对文件的管理，重点介绍文件的逻辑结构、物理结构及文件系统的构成；第 6 章介绍操作系统对设备的管理方法，主要介绍对设备的控制、分配、缓冲区的管理等；第 7 章介绍 Linux 系统的网络功能；第 8 章为现代操作系统，介绍了现代流行的 UNIX、Windows 2000 操作系统的特点、原理以及分布式操作系统的概况；第 9 章介绍操作系统的安全性。

本书由王红主编，侯刚、张凤云、杨德芳任副主编。各章主要编写人员分工如下：第 2 章、第 3 章、第 8 章由王红编写；第 1 章、第 6 章、第 9 章由侯刚编写；第 4 章由杨德芳编写；第 5 章、第 7 章由张凤云编写；参加本书编写的还有沈祥玖、王成端、陈光军、王承君、肖孟强、刘永华、宗绪锋、李禹生、安志远等。全书由王红统稿。

全书内容在介绍原理的基础上，注重理论与实践相结合。在本书的配套实训教材中，为各章提供了习题解答、内容要点复习、新增例题、习题及解答，并为各部分原理提供了一些实训内容。

本书可以作为本科，特别是应用型本科计算机专业的操作系统教材，也可以作为计算机专业技术人员的参考书。

对于本书，虽然编写小组付出了很大努力，但由于作者水平有限，出现错误在所难免，希望广大读者批评指正。

编　者

2005 年 3 月

目　　录

序
前言
第 1 章　操作系统概论 1
本章学习目标 1
1.1　操作系统的地位 1
1.2　操作系统的功能 2
1.2.1　提供人机接口 2
1.2.2　管理计算机资源 5
1.3　操作系统的发展过程 9
1.3.1　推动操作系统发展的主要动力 9
1.3.2　无操作系统的计算机系统 9
1.3.3　单道批处理系统 10
1.3.4　多道批处理系统 11
1.3.5　分时系统 12
1.3.6　实时系统 13
1.3.7　微机操作系统 14
1.3.8　网络操作系统 15
1.3.9　分布式操作系统 15
1.4　操作系统的特性 15
1.5　操作系统的体系结构 16
1.5.1　层次结构 16
1.5.2　微内核结构 16
1.6　Linux 介绍 17
1.6.1　Linux 简介 17
1.6.2　UNIX、GNU 与 Linux 18
1.6.3　Linux 的历史 20
1.6.4　Linux 的内核特征 22
1.6.5　Linux 的发展及展望 24
本章小结 26
习题一 27
第 2 章　进程管理 28
本章学习目标 28

2.1 进程的基本概念 28
2.1.1 程序的顺序执行及其特征 28
2.1.2 程序的并发执行及其特征 29
2.1.3 进程的定义与特征 30
2.1.4 进程的基本状态及转换 31
2.1.5 Linux 进程的状态 33
2.2 进程的描述 35
2.2.1 进程控制块 PCB 35
2.2.2 进程控制块的组织方式 36
2.2.3 Linux 进程的 PCB 38
2.3 进程控制 39
2.3.1 进程的创建与终止 39
2.3.2 几个相关的 Linux 系统调用 40
2.3.3 进程的阻塞与唤醒 41
2.4 进程的同步与互斥 42
2.4.1 临界资源的概念 42
2.4.2 进程的互斥与同步 43
2.4.3 锁机制 44
2.4.4 信号量机制 44
2.5 进程同步问题举例 48
2.5.1 生产者—消费者问题 48
2.5.2 读者—写者问题 50
2.5.3 哲学家进餐问题 51
2.6 进程通信 52
2.6.1 共享存储器系统 53
2.6.2 消息传递系统 56
2.6.3 管道通信系统 60
2.6.4 信号通信机制 60
2.7 线程 63
2.7.1 线程的基本概念 63
2.7.2 线程的状态与转换操作 63
2.7.3 引入线程的好处 64
2.7.4 多线程的实现 64
本章小结 65
习题二 66
第 3 章 处理机调度与死锁 68
本章学习目标 68

3.1 分级调度 68
3.1.1 调度的层次 69
3.1.2 作业与进程的关系 70
3.2 作业调度 70
3.2.1 作业调度的功能 70
3.2.2 调度算法的评价及准则 73
3.3 进程调度 74
3.3.1 进程调度的功能 74
3.3.2 进程调度的时机 75
3.3.3 进程上下文的切换 75
3.3.4 Linux 系统中进程调度发生的时机 76
3.4 调度算法 76
3.4.1 单道批处理系统的调度算法 77
3.4.2 多道批处理系统的调度算法 79
3.4.3 优先级调度算法 79
3.4.4 轮转调度算法 82
3.5 Linux 系统的调度算法 85
3.5.1 Linux 系统的进程调度策略 85
3.5.2 Linux 系统的优先级调度策略 86
3.5.3 实时进程的调度策略 88
3.6 死锁问题 88
3.6.1 死锁的概念 89
3.6.2 产生死锁的原因及必要条件 89
3.6.3 解决死锁问题的基本方法 90
3.7 死锁的预防与避免 91
3.7.1 死锁的预防 91
3.7.2 死锁的避免 92
3.8 利用银行家算法避免死锁 93
3.8.1 银行家算法中的数据结构 93
3.8.2 银行家算法的实现 94
3.8.3 银行家算法的应用 95
3.9 死锁的检测与解除 97
3.9.1 死锁检测的时机 97
3.9.2 死锁的检测 97
3.9.3 死锁的解除 99
本章小结 100
习题三 100

第 4 章　存储器管理 102
本章学习目标 102
4.1　存储器管理概述 102
4.1.1　存储器的层次 103
4.1.2　用户程序的处理过程 103
4.2　连续分配存储管理方式 106
4.2.1　单道程序的连续分配 106
4.2.2　固定分区分配方式 107
4.2.3　动态分区分配 107
4.2.4　可重定位分区 110
4.3　分页存储管理方式 111
4.3.1　页式存储管理的系统应解决的问题 111
4.3.2　分页存储管理的基本方法 111
4.3.3　两级和多级页表 114
4.4　分段存储管理方式 116
4.4.1　分段存储管理方式的基本原理 116
4.4.2　信息共享 118
4.4.3　段页式存储管理方式 118
4.5　虚拟存储器的基本概念 121
4.5.1　虚拟存储器的引入 121
4.5.2　虚拟存储器的实现方法及其特征 122
4.6　请求分页 123
4.6.1　请求分页的实现 123
4.6.2　页面置换算法 125
4.7　请求分段存储管理 127
4.7.1　请求分段的实现 127
4.7.2　段的共享与保护 130
4.8　Linux 系统的内存管理方法 131
4.8.1　Linux 的分页管理机制 131
4.8.2　虚存段的组织与管理 133
4.8.3　内存的共享和保护 135
4.8.4　内存空间管理 136
4.8.5　空闲内存管理 136
4.8.6　内核态内存的申请与释放 137
4.8.7　用户态内存的申请与释放 138
4.8.8　交换空间 139
4.8.9　页交换进程和页面换出 140

4.8.10 缺页中断和页面换入 142
4.8.11 存储管理系统的缓冲机制 142
本章小结 143
习题四 144
第 5 章 文件管理 147
本章学习目标 147
5.1 文件管理概述 147
5.1.1 文件及其分类 147
5.1.2 文件系统及其功能 149
5.2 文件的结构 150
5.2.1 文件的逻辑结构 150
5.2.2 文件的物理结构 151
5.3 文件目录管理 155
5.3.1 文件控制块和索引节点 155
5.3.2 文件目录结构 157
5.3.3 按名存取 159
5.3.4 Linux 文件目录 160
5.4 Linux 文件系统 162
5.4.1 Linux 文件的物理结构 163
5.4.2 Linux 文件系统 164
5.5 文件存储空间的管理 168
5.5.1 空闲空间表法 168
5.5.2 空闲块链法 169
5.5.3 位示图法 169
5.5.4 空闲块成组链接法 170
5.6 文件共享与安全性 172
5.6.1 文件的共享 172
5.6.2 文件的安全性 173
5.7 Linux 系统的文件操作 175
5.7.1 文件操作命令 175
5.7.2 目录操作命令 178
5.7.3 存取控制命令 179
5.7.4 磁盘存储命令 181
5.7.5 磁盘操作 182
5.7.6 常用系统调用 182
本章小结 185
习题五 186

第 6 章　设备管理 187
本章学习目标 187
6.1　I/O 系统组成 187
6.1.1　I/O 设备 187
6.1.2　设备控制器 189
6.1.3　I/O 通道 190
6.2　数据传输控制方式 192
6.2.1　程序直接控制方式 192
6.2.2　中断控制方式 193
6.2.3　DMA 方式 193
6.2.4　通道控制方式 194
6.3　中断技术 197
6.3.1　中断的概念 197
6.3.2　中断源 197
6.3.3　中断响应 197
6.3.4　中断处理 198
6.4　缓冲技术 199
6.4.1　缓冲的引入 199
6.4.2　单缓冲 200
6.4.3　双缓冲 200
6.4.4　循环缓冲 201
6.4.5　缓冲池 201
6.5　设备分配 202
6.5.1　与设备分配相关的因素 202
6.5.2　虚拟设备技术 204
6.6　SPOOLING 系统 204
6.6.1　SPOOLING 系统的简介 204
6.6.2　SPOOLING 系统的组成 205
6.7　I/O 控制过程 206
6.7.1　用户进程的 I/O 请求 206
6.7.2　设备驱动 206
6.8　磁盘 I/O 207
6.8.1　磁盘存储格式 207
6.8.2　磁盘 I/O 性能 207
6.8.3　磁盘调度 208
6.8.4　磁盘高速缓存 209
6.9　Linux 系统的设备管理 210

6.9.1 Linux 系统设备管理的特点 210
6.9.2 Linux 设备驱动程序的接口 210
6.9.3 Linux 的磁盘高速缓存 212
6.9.4 Linux 的字符设备缓冲区管理 214
6.9.5 Linux 的设备驱动 215
本章小结 218
习题六 218
第 7 章 Linux 网络基础 220
本章学习目标 220
7.1 TCP/IP 协议简介 220
7.1.1 网络概述 220
7.1.2 TCP/IP 协议 221
7.2 配置 Linux 网络 224
7.2.1 网络配置文件 224
7.2.2 手工配置 TCP/IP 网络 226
7.3 基本网络服务介绍 228
7.3.1 域名服务器（DNS） 228
7.3.2 Apache 服务器 229
7.3.3 邮件服务器 231
7.3.4 samba 服务器 232
7.4 Linux 网络安全 234
7.4.1 Linux 系统的文件安全 234
7.4.2 用户口令安全 235
7.4.3 防火墙技术 235
7.4.4 Linux 常用的网络命令 238
本章小结 241
习题七 242
第 8 章 现代操作系统实例 243
本章学习目标 243
8.1 UNIX 操作系统 243
8.1.1 UNIX 操作系统的发展 243
8.1.2 UNIX 操作系统的特点 243
8.1.3 UNIX 系统的内核结构 244
8.1.4 UNIX 系统的进程管理 245
8.1.5 UNIX 系统的内存管理 249
8.1.6 UNIX 系统的文件管理 250
8.1.7 UNIX 系统的设备管理 253

8.2 Windows 2000 操作系统 254
8.2.1 Windows 2000 概述 254
8.2.2 Windows 2000 中的进程和线程的特点 255
8.2.3 Windows 2000 中进程的实现 255
8.2.4 Windows 2000 中线程的实现 256
8.2.5 Windows 2000 的调度算法 256
8.2.6 Windows 2000 的内存管理 257
8.2.7 Windows 2000 的文件系统 257
8.2.8 Windows 2000 设备管理的特点 258
8.3 分布式操作系统 258
8.3.1 分布式操作系统的特性 258
8.3.2 进程迁移 259
8.3.3 分布式进程管理 260
本章小结 262
习题八 262
第 9 章 操作系统的安全性 **264**
本章学习目标 264
9.1 操作系统安全性概述 264
9.1.1 计算机系统安全性的内涵 264
9.1.2 操作系统的安全性 265
9.1.3 计算机系统安全性评价标准 266
9.2 操作系统的安全机制 268
9.2.1 内存保护机制 269
9.2.2 用户身份认证机制 269
9.2.3 访问控制技术 271
9.2.4 加密技术 273
9.2.5 病毒及其防御机制 273
9.2.6 监控和审计日志 274
本章小结 275
习题九 275
参考文献 **277**

第 1 章　操作系统概论

本章学习目标

本章主要讲解操作系统的基本知识，包括操作系统的地位和作用、操作系统的功能、操作系统的发展过程以及 Linux 操作系统的一些基本知识。通过本章的学习，读者应该掌握以下内容：

- 掌握操作系统的功能和地位
- 通过学习操作系统的发展过程，掌握各种不同类型操作系统的特点及应用
- 掌握操作系统的特征
- 理解操作系统的结构，掌握层次结构的操作系统结构模型

1.1　操作系统的地位

计算机系统由硬件和软件构成，硬件是计算机系统的物质基础，完成基本操作，软件通过对硬件基本功能的排列实现人们所需的高级功能，如学习、游戏、娱乐、办公，人们在高层通过软件来使用计算机系统完成各种工作，软件是计算机系统的灵魂。正如人们弹奏钢琴一样，作为硬件的钢琴提供了固定数量的能发出基本音符的琴键，通过演奏者对这些固定数量的琴键的排列弹奏，就能奏出无限多的乐曲来。

计算机系统是分层次的，最低层是未配置任何软件的硬件裸机，硬件之上是软件，软件又分为若干层次，最低层是操作系统，如图 1-1 所示。

操作系统是覆盖在裸机上的第一层软件，它直接控制、管理各种硬件资源。在裸机上安装了操作系统后，就为其他软件和用户提供了工作环境。操作系统要为人们有效地使用计算机提供用户接口，首先要提供作业控制接口，使用户能通过终端输入一定命令来控制作业流程，如启动一个应用程序，通过这个接口用户还可以完成一些基本操作，如文件操作、显示或设置系统日期、时间等；而且操作系统还为应用开发人员提供了系统功能调用，提高了应用开发效率。操作系统还要对系统资源进行统一管理，使各并发进程能按一定原则合理共享系统资源，并在保证各并发进程顺利运行的基础上提高资源利用率。所以操作系统是整个计算机系统的控制管理中心。

各类应用用户使用相应的应用软件完成各种工作，如财务管理人员使用财务管理软件进行财务管理工作，办公室人员利用文字表格处理软件处理各类文件和表格，学生使用教学课件学习某门课程，电脑游戏爱好者在电脑上使用游戏软件玩游戏等。计算机已经应用到社会

生活的各个领域，对于各种应用都要有相应的应用软件来提供支持。计算机系统就是为人们提供各类服务的，应用软件由各应用用户直接使用，所以处在计算机系统的最高层。

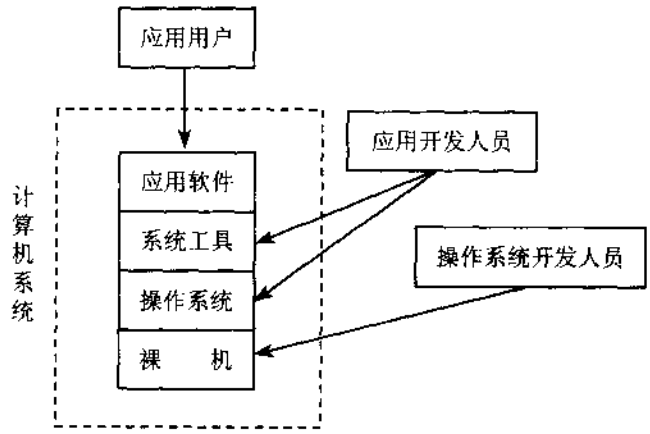


图 1-1　计算机系统层次结构

应用软件由应用开发人员开发，为了提高开发效率，采用一些开发工具，如高级语言编程工具（包括编辑工具、编译工具、连接工具、调试工具）软件，还可通过系统调用命令调用操作系统的功能模块。

开发工具为应用软件的开发搭建了一个平台，为方便用户开发应用软件，许多开发工具对系统功能调用进行了包装，提供给用户更高级的功能调用，如 Turbo C 中的库函数、Visual C 中的 MFC，这些开发工具在实现用户所要求的某些功能时，不由自身实现，而是调用操作系统的功能。

应用软件和开发工具都工作在操作系统提供的运行环境中，它们的运行由操作系统进行管理，由操作系统为它们分配所需的资源，并由操作系统对 I/O 设备进行控制。

1.2　操作系统的功能

操作系统是计算机系统中具有一定功能的软件系统。操作系统的目的是方便用户使用计算机系统和提高计算机系统资源利用率。

1.2.1　提供人机接口

用户是通过操作系统提供的人机接口来使用计算机的。操作系统为用户提供了两种接口。

1. 作业控制级接口

作业是用户上机所作的一系列顺序相关的工作。一道作业由若干顺序相关的作业步构成。例如，上机编程要经历如下步骤。

编辑：录入用某种编程语言书写的源程序，建立源程序文件，如用 C 语言写的源程序。

这个作业步需要使用编辑软件完成。源程序文件是无格式的文本文件，只要能够编辑文本文件的编辑程序就可以编辑源程序，建立源程序文件。用 Turbo C 编程时，采用了一个集成编程环境，其中就集成了一个文本编辑器，可以直接利用其编辑源程序，而不需要再启动另外的编辑器。编辑汇编语言源程序则必须使用独立的文本编辑器，如 DOS 环境下的文本编辑器 EDIT。

编译：把源程序翻译成由机器指令构成的目标程序。这个作业步需要由编译软件完成。Turbo C 的集成编程环境提供了一个编译器，可以在编辑完源程序后，在同一环境中通过菜单或快捷键启动编译器，对源程序进行编译。汇编语言源程序则需要用独立的编译软件进行编译。

连接：把编译所得的各个目标模块和程序中调用的库函数连接装配在一起，构成一个完整的、可装入执行的目标程序文件——可执行文件。

完成一个编程任务，必须经过以上 4 个作业步，这些作业步是相互关联、顺序实施的。作业步之间的关系表现为：

（1）每个作业步运行的结果是产生下一个作业步所需的文件，如图 1-2 所示，图中各程序都是以文件的形式存放在外存中的。

（2）一个作业步能否开始执行依赖于前一个作业步是否顺利完成。如图 1-2 所示，只有当编辑完源程序后才能对源程序进行编译，编译成功后才能进行连接，连接成功后才能装入程序并运行。

（3）各作业步之间的关系不一定是单向的，而可以从当前作业步返回上一个作业步。如图 1-2 所示，程序的编辑、编译、连接、运行不一定能一气呵成，中间可能会出现错误。如果在编译时出现错误（即编译程序对源程序检查后发现源程序中有语法错误），则需要返回到编辑步骤，对源程序进行修改，然后再重新编译，若仍存在错误，还要返回编辑步骤修改源程序，如此反复进行，直至编译通过。如果在运行时出现错误，一般是程序中存在算法错误，此时需要返回到编辑步骤，修改源程序中的算法，然后重新编译、连接、运行。

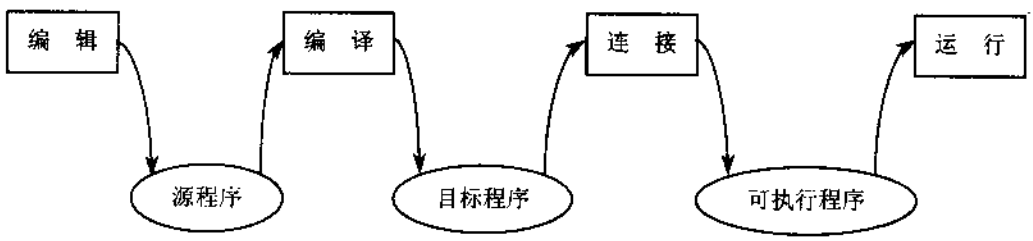


图 1-2　作业示例

以上作业的工作流程应由用户按自己的需求进行控制，因此要提供用户控制作业工作流程的手段，这是由操作系统提供的，称为作业级接口。作业级接口由一组用户可直接使用控制作业运行的命令和命令解释器构成，例如，在 DOS 系统中有内部命令和外部命令，有命令解释器 command.com。该接口又可进一步分为联机用户接口和脱机用户接口。

（1）联机用户接口。由一组键盘字符命令（或鼠标命令）和命令解释器组成，使用

户可以联机交互方式使用计算机。用户每次键入一个合法命令（解释器能执行的命令），启动一个作业步，一个作业步运行完毕后，再键入下一个命令名，启动下一个作业步。在一个作业步结束后，若发现错误，可以由用户修正错误，然后重新启动该作业步。用户可根据作业运行情况随时进行作业步的调整。

联机用户接口又可分为键盘字符命令接口和图形接口。

键盘字符命令接口的操作命令为键盘字符形式，通过键盘下达命令。要求用户能熟记各个命令的名字和格式，并严格按照规定的格式输入命令，很不方便。如 DOS 提供的键盘字符命令接口，根据命令对应程序的所在位置，分为内部命令和外部命令，内部命令对应的程序在操作系统启动时自动装入内存，外部命令对应的程序以可执行文件的方式存放在外部存储介质上。

图形接口采用图形化的操作界面，用形象、直观、容易识别的各种图标表示功能、程序和文件。用户主要通过鼠标下达命令，进行各种操作，不需要记忆键盘字符命令和格式，使用户使用计算机更方便且生动有趣。许多操作系统都提供了图形接口，如 IBM 公司的 OS/2、Apple 公司的 Macintosh 以及 Microsoft 公司的 Windows。

（2）脱机用户接口。操作命令的形式为作业控制语言，用户以脱机批处理方式使用计算机。用户对作业流程的控制意图是利用作业控制语言书写成一份作业说明书来表达的。上机时，用户将作业控制说明书交给系统，系统逐条解释执行说明书中的命令。在这种方式下，用户一旦提交了作业，作业流程就由操作系统根据作业控制说明书自动控制，用户无法干预该作业的运行。因此，必须事先设计好作业流程，还要预测作业运行过程中可能出现的错误，并给出发生错误时的处理方法。

有的操作系统同时提供以上两种作业控制接口。

联机用户接口和脱机用户接口各有优点和缺点。联机用户接口使用户可以灵活方便地控制作业流程，及时处理作业运行过程中出现的错误，但系统的运行效率低，因为作业运行过程中作业步的衔接依靠人工干预控制，操作速度较慢，占用了很多时间。脱机用户接口使作业自动运行，作业步之间自动过渡，没有人工干预操作，提高了作业运行效率，但不能灵活地控制作业流程，发现错误不能及时处理，只能等整个作业运行完才能处理作业中存在的错误，这使得作业上次的运行结果错误而失去意义，只能修正错误后重新运行该作业，而且编写详尽的作业控制说明书很困难。

2. 程序级接口

操作系统提供的程序级接口由一组系统功能调用命令以及完成这些命令的程序模块组成。为方便用户编程，提高编程效率，规范编程，操作系统提供了完成某些通用功能的程序，供用户在开发应用程序时调用。不同的操作系统提供了不同的系统功能调用以及调用方式，如 DOS 的系统功能调用主要是进行硬件驱动，以软中断 INT 21H 的方式提供。Windows 中的系统功能调用要比 DOS 丰富，且层次高，不只局限于硬件驱动，以可在编程语言中使用的应用编程接口函数的方式提供，称为 API—Application Programming Interface 。使用 Windows 的 API 函数可以提高编程效率，并规范 Windows 环境下的编程，如可开发具有统一风格的应用程序窗口界面，使软件用户很快熟悉该软件的窗口界面而不

必重新学习。

用 DOS 环境中的 Turbo C 编程时，经常用到库函数。Turbo C 编程工具提供了完成某些特定功能的函数供用户编制应用程序时调用，如标准输入函数 scanf()、标准输出函数 printf()。有些库函数就是利用 DOS 系统功能调用来实现的，对 DOS 系统功能调用进行了封装，构成了更便于使用的库函数。Visual C 中的 MFC（微软基本类库）是利用 API 函数实现的。在 Linux 或 UNIX 系统中，用户通过 C 语言进行系统调用。

1.2.2　管理计算机资源

计算机系统中的资源包括硬件资源和软件资源。硬件资源有处理机、存储器、外部设备；软件资源有程序和数据。

现代计算机系统一般采用多任务并发执行方式，而且有的计算机系统由多个用户同时联机使用，这使得计算机的资源不是由一个程序在运行时独占使用，而是由多个并发运行的程序共享使用。如果由各并发运行的程序自己决定如何使用资源，则会各行其是，造成冲突、混乱，使系统无法顺利地高效运行。因此，要有一个地位高于各应用程序之上的软件来进行自动的、统一的管理，这个软件就是操作系统。操作系统的目标之一就是统一管理分配计算机系统资源，在保证各并发执行的应用程序顺利运行的前提下提高资源利用率。

1. 处理机管理

处理机的任务是运行程序，程序在某个数据对象上的一次运行过程称为进程，所以处理机管理又称为进程管理。

单处理机系统中，程序有两种运行方式：单道程序顺序执行和多道程序并发运行。

单道程序顺序执行：要执行的多个程序按一定次序依次执行，一个程序运行完毕才能运行下一个程序，即在一个程序运行期间不插入运行其他程序。这种运行方式的优点是实现简单，不需要在多个进程之间进行转换；缺点是资源利用率低，因为在系统运行的整个过程中，始终是一道程序独占系统全部资源，一般一道程序不会同时进行计算和输入输出操作，这使得处理机和设备串行工作，在处理机忙时设备空闲，在设备忙时处理机空闲，并且，这样的系统无法提供多用户同时联机使用方式。

多道程序并发执行：在内存中同时存放多道程序，按一定策略调度多道程序交叉运行，形成“微观上串行、宏观上并行”的情况，使得处理机和设备可以并行工作，当某个进程进行输入输出操作时，可以同时有另一个进程在处理机上进行计算。从直观上看这种方式肯定比单道程序顺序执行方式的资源利用率高，当然这取决于采用何种处理机调度策略。再者，多道程序系统可以提供多个用户同时联机操作方式，一台主机可以同时连接若干用户终端，同时若干用户可分别通过自己的终端使用主机系统。

我们在此讨论的是单处理机系统中多道程序并发运行方式下的处理机管理。具体包括如下。

（1）处理机调度。要在单处理机系统中并发运行多道程序，必须按照一定策略对处理机进行调度，就像在一条铁路上运行多列火车一样，需要进行调度，要决定在某个时刻把处理机分配给哪个进程进行计算操作，这是处理机管理的核心任务。

（2）进程控制。进程是程序的一次动态运行过程，在其生存期内从产生到消亡经过了一系列状态的转换。在多道程序并发运行的系统中，通常不会让一道程序独占计算机全部资源不间断地运行，而是让多个进程交替运行。这样进程在其生存期内肯定有多种状态，至少有运行和不运行两种状态。进程的状态数量是操作系统按照一定的管理策略设置的，进程状态转换是操作系统实施进程管理的一个基本操作。操作系统提供了一些原语对进程状态的转换进行控制。原语是指在运行过程中不可中断的、完成某一功能的一段程序，作为操作系统的内部功能模块存在，之所以称为“原语”，是取了人们原来认为原子不可分割之意。

（3）进程通信。在多道程序环境下，可以由系统为一个应用程序建立多个进程，这些进程相互合作，完成某一共同任务，它们之间要交换信息——进程通信。为保证进程之间正确通信，操作系统提供了一系列通信原语供应用进程调用。

（4）进程同步。

在多道程序系统中，多个并发进程处在同一运行环境中，必然存在某种联系，如进程之间的资源共享和进程之间的协作。操作系统必须采用一定策略来处理并发进程之间的制约关系，使各进程能顺利运行，即使各进程同步协调运行。操作系统中设置了同步机制来完成此功能。

2. 存储管理

计算机系统采用了冯·诺依曼提出的存储程序原理，即把要运行的程序一次性存放在存储器中，然后由处理机自动从存储器中依次取出程序指令运行，处理机的运行过程就是不断地取指令、执行指令循环往复的过程，每次取一条指令，执行一条指令。存储器是计算机系统中的重要资源，与处理机一起称为计算机系统中的主机。因此，程序的运行机构不只是处理机，而是由处理机和存储器构成的主机。

在多道程序环境中，要在内存中同时存放多道程序，必须对内存进行合理管理以保证程序的顺利运行，并提高内存的利用率。

操作系统提供了如下存储管理功能。

（1）内存分配。内存分配的任务是为每道程序分配一定的存储空间。往往会出现多道程序对内存的需求总和超过实际内存空间，因此，制定分配策略时应以提高内存利用率为目标。

（2）地址转换。一个应用程序经过编译后，形成若干个目标程序，这些目标程序经过连接而形成可装入程序（可执行程序）。在可执行程序中，有一些指令中包含内存地址码，要访问该地址处存放的数据或指令，如内存单元和CPU寄存器之间的数据传送指令中包含数据的内存地址码，转移指令中包含指令的内存地址码。将程序装入内存之前，无法确定程序在内存中的实际存放位置，因此必须建立一个虚拟内存地址空间，将整个程序放在这个虚拟内存地址空间中，每个数据和指令都有一个惟一确定的虚拟内存地址，这个地址空间又称为逻辑地址空间，相对地，称实际内存地址空间为物理地址空间。

程序必须装入内存才能被CPU访问并执行，程序指令的操作对象——数据也必须装入内存才能被程序访问，程序和数据装入内存后占据一定的物理内存地址空间，程序的逻辑

地址空间往往和物理地址空间不重合，而程序最终是在物理地址空间中运行，即访问程序中的指令和数据时是按其所在实际内存单元的地址进行的，所以要正确地访问数据和指令，必须把指令中的逻辑地址转换成相应的物理地址，这一操作称为地址转换，又称为重定位。

（3）内存保护。内存保护的任务是确保每道用户程序都在自己的内存空间中运行，互不干扰，即不允许用户程序访问操作系统的程序和数据（只能通过系统调用访问操作系统），也不允许访问（存取）其他程序的存储空间。操作系统要提供某种有效的内存保护机制，当进程访问内存越界时能及时发现并进行处理。内存保护不排斥对程序代码和数据的共享，此时把共享的代码和数据作为主调进程的一个组成部分，同一程序代码和数据共享的进程所占用的内存空间有部分重叠。

（4）内存扩充。在多道程序环境中，内存资源比较紧张，往往会出现多道程序对内存的需求总和超过实际内存容量的情况。因此在采用合适的内存分配策略以提高内存利用率的基础上，应采用内存扩充技术为多道程序运行提供充足的内存空间。这里说的内存扩充是从逻辑角度而言的，不是指添加物理内存，扩充物理内存容量。既然不增加物理内存，那么就要从逻辑角度扩充内存，方法是在一定内存空间中运行对内存需求超过此空间容量的进程。有两种方法：一是进程的整体交换，即将某些在以后一段时间内不能运行的进程的代码和数据暂时撤出内存，放到外存中，将当前急需运行的进程的代码和数据调入内存，对于换到外存中的进程，当其运行条件具备时，再将其调入内存；二是进程的部分装入，给进程分配少量内存，先装入一部分开始运行，当要访问不在内存中的部分时，再将其调入内存。这两种方法并没有扩充物理内存，但其作用相当于扩充了物理内存。

3. 设备管理

设备管理的任务是接受用户程序提出的 I/O 请求，为用户程序分配 I/O 设备；使 CPU 和 I/O 设备并行操作，提高 CPU 和 I/O 设备的利用率；提高 I/O 速度；方便用户程序使用 I/O 设备。为完成以上任务，操作系统的设备管理子系统应具有设备分配、缓冲管理、设备驱动、设备无关性等功能。

（1）设备分配。根据用户程序的 I/O 请求，为之分配所需的设备。如果 I/O 设备和 CPU 之间还存在设备控制器和 I/O 通道时，还要为分配出去的设备分配相应的控制器和通道。

为实现设备分配，系统中应配置设备控制表、控制器控制表等数据结构，用于记录设备及控制器的标识符和状态，以供设备分配时参考。进行设备分配时，应针对不同设备类型而采用不同的设备分配方式。对于独占设备的分配，还应考虑该设备被分配出去后，系统是否安全。设备用完后，应立即加以回收。

（2）缓冲管理。进程在运行过程中要和输入输出设备之间传输数据，在进程的数据存储区和 I/O 设备控制器之间直接传输数据效率较低或无法进行，因此操作系统提供了缓冲技术。缓冲管理的任务是解决 CPU 和外设速度不匹配的矛盾，使它们能充分并行工作，从而提高 CPU 和 I/O 设备的利用率，最终达到提高系统吞吐量的目的。

（3）设备驱动。设备驱动程序实现 CPU 与设备控制器之间的通信。由 CPU 向设备控制器发出 I/O 指令，由设备控制器驱动相应设备进行 I/O 操作；当 I/O 操作完成时，设备控制器向 CPU 发出中断信号，由相应的中断处理程序进行处理。

处理过程是设备处理程序首先检查 I/O 请求的合法性、了解设备的状态是否空闲、了解有关传递参数以及设置设备的工作方式。然后，向设备控制器发出 I/O 命令，启动 I/O 设备完成指定的 I/O 操作。最后及时响应由控制器发来的中断请求，并根据该中断请求的类型，调用相应的中断处理程序进行处理。对于设置了通道的计算机系统，设备处理程序还应能够根据用户的 I/O 请求，自动构成通道程序，由通道程序控制 I/O 设备完成具体的 I/O 操作。

（4）设备无关性。设备无关性又称设备独立性，即用户编写的应用程序与实际使用的物理设备无关。用户编写的应用程序中不直接指定使用哪台具体的物理设备，而是使用操作系统提供的逻辑设备，然后由操作系统把用户程序中使用的逻辑设备映射到具体的物理设备，实施具体的 I/O 操作。这样做的一个明显好处是用户应用进程的运行不取决于某台具体物理设备的状态，而由操作系统为其分配一台合适的设备完成 I/O 操作，这样会避免出现有设备可用但进程却无法运行的情况。

4. 文件管理

计算机系统中的软件资源（程序和数据的集合）不是一次性用品，而是要反复利用的，因此要永久保存（相对于内存的暂时存储而言）起来，如银行中的存贷款数据、学校的学籍管理软件和学籍数据等。软件资源以文件的形式存放在外部存储介质中，供用户反复使用。操作系统中对文件进行管理的子系统称为文件系统，文件系统的任务是为用户提供一种简便、统一的存取和管理文件的方法，对用户而言，按名存取是一种简便的存取文件的手段；实现文件的共享；维护文件的秘密和安全。文件管理有如下具体功能。

（1）文件存储空间的管理。为新文件分配所需的外存存储空间，回收释放的文件存储空间。进行文件存储空间的分配和回收时，要考虑到提高外存空间的利用率和提高文件的存取速度。

（2）目录管理。为了能方便地在外存中找到所需文件，要为外存中存放的文件建立目录，每个文件对应一个目录项。目录项包含文件名、文件属性、文件在外存中的存放位置等用户和操作系统所需信息。目录管理的主要任务是建立外存中文件的目录结构，实现用户程序对文件的“按名存取”。文件目录项要按一定结构组织起来，以便于操作系统检索。在 UNIX/Linux 中采用了树型目录结构。

（3）文件操作。文件操作包括文件的创建、删除、打开、关闭、读、写等，其实是一组文件系统功能调用，在用户程序中对文件进行操作时，可以调用文件系统提供的这些文件操作功能。

（4）文件的存取权限控制。为了防止系统中的文件被非法窃取和破坏，操作系统提供了文件的存取权限控制功能，以防止未被授权的用户存取文件，或以不正确的方式存取文件。当用户对某一文件操作时，首先检查其对该文件的操作权限，如果具有对该文件的此种操作权限，则允许该用户存取该文件，否则拒绝对该文件的此种操作。

根据以上所述操作系统的功能，我们可以给操作系统下一个描述性的定义，即操作系统是一个软件系统，它控制和管理计算机系统内各种硬件和软件资源，提供用户与计算机系统之间的接口。

1.3　操作系统的发展过程

1.3.1　推动操作系统发展的主要动力

1. 不断提高计算机资源利用率的需要

在计算机系统发展的初期，计算机系统特别昂贵，人们迫切需要提高计算机系统中各种资源的利用率。随着计算机的普及，硬件价格下降了很多，但是计算机的资源仍然紧缺，特别是内存资源，所以资源利用率仍然是操作系统关注的重要问题。

2. 方便用户

计算机系统要方便用户使用，主要体现在提供友好的用户操作界面；提供丰富实用的系统功能调用；提供人机交互的操作方式。

3. 器件的不断更新换代

计算机的制造器件在不断发展，由电子管发展到晶体管、集成电路、大规模集成电路，使得计算机性能不断提高，规模不断扩大。操作系统是基于硬件系统的，直接对硬件进行管理和控制，所以计算机制造器件的发展必然推动操作系统的发展。

4. 计算机体系结构的不断发展

操作系统基于硬件体系结构，所以硬件体系结构的发展必然推动操作系统的发展。例如，当计算机由单处理机系统发展为多处理机系统时，操作系统也就由单处理机操作系统发展为多处理机操作系统；出现了计算机网络后，就有了网络操作系统。

1.3.2　无操作系统的计算机系统

世界上公认的第一台数字电子计算机于 1946 年问世，人们通常按照计算机元器件的演变把计算机的发展过程分为 4 个阶段。

第 一代：1946 年～1958 年　电子管
第二代：1958 年～1964 年　晶体管
第三代：1964 年～1974 年　小规模集成电路
第四代：1974 年至今　大规模集成电路

现在计算机正向巨型化、微型化、网络化、智能化方向发展。

在第一代计算机时期，构成计算机的主要元器件是电子管，其运算速度较慢（只有几千次每/秒）。此时，计算机由主机（运算器、控制器、内存）、输入设备（纸带输入机、卡片阅读机）、输出设备（打印机）和控制台组成。此时，人们采用手工方式使用计算机，用户一个挨一个地轮流使用计算机。每个用户的工作过程大致是先把程序纸带（或卡片）装到输入机上，然后启动输入机把程序和数据输入计算机存储器，接着利用控制台开关启动程序开始执行。计算结束，用户取走打印出来的结果，并卸下纸带（或卡片）。在这个过程中，需要人工装卸纸带、人工控制程序运行。手工操作速度相对计算机的运行速度而言是很慢的，因此使用计算机完成某一工作的整个过程中，手工操作时间占了很大的比例，而

计算机运行时间所占比例较小，这就形成了明显的人机矛盾，致使计算机资源利用率很低，从而使计算机工作效率很低。在早期计算机运行速度较慢时，这种状况还是可以容忍的。

当计算机进入第二代——晶体管时代后，计算机的速度有了很大提高，这使得人机矛盾显得很突出，严重制约了计算机的工作效率。在当时，计算机非常少，并且贵重，人们迫切希望能高效率地使用计算机。为了解决这种矛盾，就要设法减少手工操作时间，因此人们研制出了实现作业自动过渡的批处理系统。

1.3.3 单道批处理系统

单道批处理系统在当时称为监督程序，是操作系统的雏形。监督程序常驻内存，在它的控制下，实现了作业的自动过渡，从而去掉了原先作业过渡时的手工操作。此时，出现了汇编语言、高级语言编程工具，每一种语言编译程序（如汇编语言或某种高级语言的编译程序）、实用程序（如连接程序）都作为监督程序的子例程，用到它们时由监督程序进行调用。

早期的批处理分为联机批处理和脱机批处理两种。

1. 联机批处理

操作员把一批作业装到输入设备上（纸带输入机/卡片阅读机），然后由监督程序控制，将这批作业输入到磁带上，在监督程序的控制下，使这批作业一个接一个的连续执行，直至所有作业运行完毕。联机批处理的模式如图 1-3 所示。

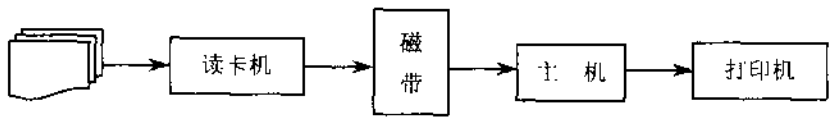


图 1-3 联机批处理

2. 脱机批处理

联机批处理系统实现了作业的自动过渡，同手工操作相比，计算机的运行效率提高了。但是，联机批处理系统中，作业的输入输出是联机的，也就是说作业从输入机到磁带，从磁带调入内存，输出打印结果，都是中央处理机直接控制（由处理机运行相应程序完成）。在这种操作方式下，作业进行输入输出时，CPU 未做有效的计算工作，而且由于 I/O 设备的速度远远落后于 CPU 的运行速度，在作业的整个处理的时间内，I/O 操作占用了较大的比例，而计算机的有效运行时间占用比例较小，从而导致处理机的利用率较低。为了克服这一缺点，在批处理系统中引入了脱机输入输出技术。

脱机批处理系统由主机和卫星机组成。卫星机又称外围计算机，不与主机直接连接，只与外部设备打交道。其工作方式是，卫星机把纸带/卡片上的作业输入到磁带上，当主机需要执行作业时，就把输入磁带连接到主机上，主机从输入磁带上把作业调入内存，并予以执行，作业完成后，主机把计算结果送到输出磁带上，然后由卫星机负责把输出磁带上的信息输出打印。这使主机摆脱了对慢速输入输出设备（纸带/卡片输入机、打印机）的控

制，可以用更大的时间比例进行有效的计算工作，更充分地发挥高速计算能力，提高了利用率。由于主机和卫星机可以并行工作，因此脱机批处理系统的处理能力大大提高。脱机批处理的模式如图 1-4 所示。

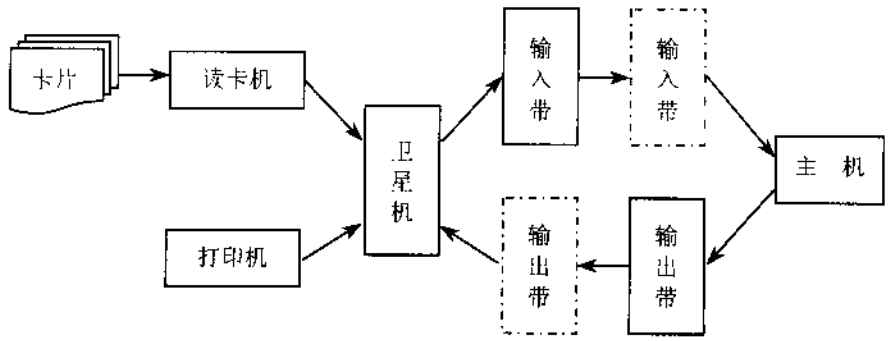


图 1-4　脱机批处理

批处理系统是在解决人机矛盾、中央处理机高速度和 I/O 设备的低速度这一对矛盾的过程中发展起来的，当时的目标是提高计算机的工作效率，它的出现促进了软件的发展，出现了监督程序、汇编程序、编译程序、连接程序等。

1.3.4　多道批处理系统

单道批处理系统中任何时刻只有一道作业在内存中，在一道作业的运行过程中，输入输出和计算操作是串行的，因此导致 I/O 设备和 CPU 串行工作，从而总有空闲资源，I/O 设备工作时 CPU 空闲，CPU 工作（指进行计算工作）时 I/O 设备空闲。即使在脱机批处理系统中，内存和输入输出磁带间的数据传输与 CPU 的计算工作也是串行的。

为了进一步提高资源利用率，最终提高系统吞吐量（系统在单位时间内完成的总工作量），在 20 世纪 60 年代中期引入了多道程序并发执行技术，从而形成了多道批处理系统。多道程序并发执行的基本思想是，内存中同时存放多道程序，在操作系统的控制下交替执行。在多道批处理系统中，用户提交的作业先存放在外存中，并排成一个队列，称为后备队列，然后由作业调度程序按一定策略从后备队列中选择若干作业调入内存，使它们并发运行，从而共享系统中的各种资源，提高资源利用率，最终提高系统吞吐量。

虽然早在 20 世纪 60 年代就出现了多道批处理系统，但它至今仍然在使用，许多大、中、小型机上都配置了这种操作系统。

多道程序并发执行系统的特征如下。

（1）多道性。在内存中同时驻留多道程序，分别为它们创建进程。

（2）调度性。单处理机系统中，每个时刻只能运行一道程序指令，所以同时在内存中的多道程序不能同时在一个 CPU 上运行，必须进行调度，即采用合理的调度策略使多道程序并发执行。

（3）宏观上并行，微观上串行。在单处理机系统中，同时处于内存中的多道程序在

微观上交替地占有 CPU 运行，是串行的，而在操作系统的调度下，用户感觉多道程序在并行运行。

（4）异步性。内存中的多道程序各自开始执行的时间、结束时间不由其进入内存的次序决定，在操作系统的统一调度下，多道程序以不可预知的时间开始运行，并以不可预知的速度运行，不可预知其结束时间。

1.3.5 分时系统

多道批处理系统充分地提高了计算机资源利用率和系统吞吐量，但是缺少人机交互能力，即用户把作业提交给计算机系统后，就完全脱离了自己的作业，不能干预作业的运行，因此不能及时修正作业运行过程中出现的错误，只有当作业运行结束后才能脱机修正错误，因此使用不方便。每当编写好一个程序时，都需要上机进行调试。由于新编写的程序难免有些错误或不当之处，因此希望能够进行人机交互以便能及时地修改错误，即用户希望可以随意干预、控制自己作业的运行流程。

由此开发出了交互式分时操作系统。在分时系统中，一台主机可以连接若干终端，如图 1-5 所示，每个用户可以通过终端与主机交互，方便地编辑和调试自己的程序，向系统发出各种控制命令，系统及时地响应用户的请求，输出计算结果以及出错、告警、提示信息。

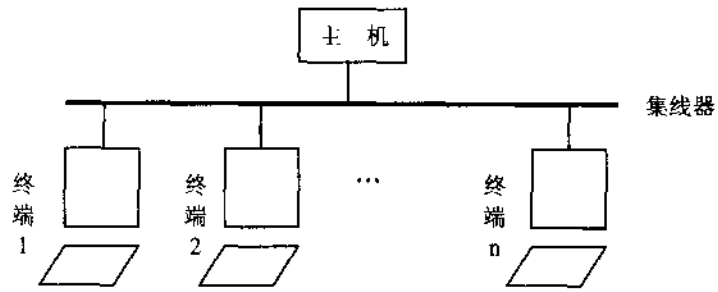


图 1-5 分时系统

在分时系统中，虽然若干用户通过各自的终端共享一台主机，但在操作系统的管理下，每个用户都感觉自己在独占一台主机。分时系统采用的策略是基于主机的高速运行，分时为终端用户服务，即主机按一定次序轮流为各终端用户服务，每个用户一次仅使用主机很短的一段时间（称为时间片，毫秒级），在分得的时间片内若没有完成工作则暂时中断，将处理机分配给下一个用户。虽然一个用户使用主机时其他用户处于等待状态，但等待的时间很短，用户感觉不到，从而每个用户的各次请求都能得到快速响应，给每个用户的印象是他独占一台计算机。

分时系统强调的是人机交互，但并没有忽略系统资源利用率和运行效率。用户在交互使用计算机系统时，手工操作终端的时间占用了较大比例，而且使用过程中会不时地停下来进行思考。如果由一个用户独占计算机系统，那么系统的资源利用率和运行效率很低。

为了使系统既能提供人机交互性，又能高效运行，分时系统采取了多用户分时共享一台主机的策略。当一个用户中间停顿时，其他用户使用主机，从而使主机高效运行。

分时系统具有以下特征。

（1）多个用户同时联机操作。同一台主机同时连接多个终端，各用户独占一个终端，通过终端使用主机。

（2）各用户独立。在宏观上，多个用户通过各自的终端同时使用一台主机，即一台主机同时为若干用户服务，而各用户之间互不干涉。

（3）交互性。各用户通过终端联机以交互方式使用主机。交互式与批处理相对，交互式使用计算机是指用户可以随意干预、控制作业的运行流程，这需要操作系统提供一组人机交互命令，如 DOS 中的内部命令和外部命令，UNIX/Linux 中的 shell 命令。

1.3.6　实时系统

虽然多道批处理系统和分时系统已经获得令人满意的资源利用率和响应时间，使计算机的应用范围日益扩大，但它们不能满足以下两个领域的需要。

1. 实时控制

当把计算机用于生产过程的控制，形成以计算机为中心的控制系统时，系统要求能实时采集现场数据，并对所采集的数据进行及时处理，进而自动控制相应的执行机构，使某些（个）参数（如温度、压力、方位等）能按预定的规律变化。类似地，也可将计算机用于武器的控制，如火炮的自动控制系统、飞机的自动驾驶系统，以及导弹的制导系统等。通常把要求进行实时控制的系统称为实时控制系统。

2. 实时信息处理

通常，把要求对信息进行实时处理的系统称为实时信息处理系统。该系统由一台或多台主机通过通信线路连接成百上千个远程终端，计算机接收从远程终端发来的服务请求对数据进行检索和处理，并及时将结果反馈给用户。典型的实时信息处理系统有飞机订票系统、情报检索系统。

我们把实时控制系统和实时信息处理系统称为实时系统。所谓实时，是指计算机能及时响应外来事件，并快速地处理该事件，在被控对象允许的时间范围内作出快速反应。实时系统对响应时间的要求比分时系统高。

实时系统具有以下特征。

（1）及时性。系统必须能及时响应外部实时信号，响应的时间间隔要足以能够控制发出实时信号的环境。

（2）可靠性。由于实时系统的应用环境特殊，任何软硬件故障都可能带来严重的后果，因此，必须采取相应的软硬件措施，以保证系统高度可靠。常用的硬件措施如采用双工机制，即准备两台功能相同的计算机，将其中一台作为主机，另一台作为后备机，后备机与主机并行工作，但不产生控制输出，若主机发生故障，后备机立即代替主机继续工作，保证系统不间断运行。

1.3.7 微机操作系统

随着大规模集成电路的发展，产生了微处理器，采用微处理器构造的计算机称为微机。微机是面向个人的，所以又称个人计算机，区别于连接若干终端，由若干用户共享的巨型、大型、中型、小型计算机。由于个人计算机的特殊应用要求，需要专门配置微机操作系统。微机操作系统的主要任务是使个人用户方便地使用计算机，对资源利用率考虑得较少。

微机操作系统有很多，比较流行的有 DOS、Windows、UNIX/Linux。

微机操作系统按照用户数、任务数可作如下划分。

1. 单用户单任务操作系统

单用户单任务是指，只允许一个用户上机，要运行的多个程序须按一定次序依次执行，不能交替执行。这是最简单的微机操作系统，代表性产品是 CP/M 和 MS-DOS。

（1）CP/M。CP/M 是 Control Program Monitor 的缩写，是 1975 年由 Digital Research 公司开发的 8 位微机操作系统，配置在以 Intel 8080、8085、Z80 芯片为 CPU 的微机上。

（2）MS-DOS。该操作系统是由微软公司在 CP/M 基础上开发的，配置在 IBM 公司生产的 PC 机及其他公司生产的 PC 兼容机上。MS-DOS 对 CP/M 进行了改进，增加了许多内部命令、外部命令，提供了性能优良的文件系统。由于 IBM-PC 机及其兼容机的畅销，MS-DOS 成为事实上的单用户单任务操作系统的标准。

2. 单用户多任务操作系统

单用户多任务是指，只允许一个用户上机，但可以并发执行多道程序，从而充分利用系统资源，满足用户同时执行多个任务的需求，如一边打字一边听音乐，代表性产品是 OS/2 和 Windows。

（1）OS/2。1987 年 4 月，IBM 公司推出了 OS/2，其最初版本 OS/2 1.X 是针对 80286 开发的，后来针对 80386 和 80486 开发了 OS/2 2.X 版本。

（2）Windows。1990 年微软公司推出 Windows 3.0，其友好的图形用户界面和易学易用的特点使其很快推广开来。1992 年推出 Windows 3.1，提供了 386 增强模式，提高了运行速度，增强了功能。1993 年推出的 Windows NT 提供了很强的网络支持功能。1995 年推出 Windows 95，1998 年推出 Windows 98，之后又推出了 Windows 2000、Windows ME、Windows XP、Windows 2003。事实表明，Windows 已经成为单用户多任务微机操作系统的主流。

3. 多用户多任务操作系统

微机是面向个人用户而开发的，所以一般由单个用户使用，配置单用户操作系统。但这并不意味着微机不可由多个用户同时联机使用，特别是现在的微机与小型机的差距已经很小，只要在微机上配置多用户操作系统就可以使微机同时为多个用户服务。具有代表性的产品是 UNIX 和 Linux。

UNIX 是由美国 AT&T 公司的 Bell 实验室开发的，至今已有 20 多年历史。最初，UNIX 配置在 DEC 公司生产的小型机 PDP 上，后来被移植到各种机型上。UNIX 操作系统是目前惟一能够在从微机到大型机等各种机型上运行的操作系统，是当前最流行的多用户多任

务操作系统。

Linux 是一个类 UNIX 操作系统，最初由芬兰赫尔辛基大学的 Linus 开发。

1.3.8　网络操作系统

随着计算机应用的推广出现了计算机网络。计算机网络是在计算机技术和通信技术高度发展的基础上，将这两者相互结合而产生的。为了实现计算机之间的数据通信和资源共享，把分布在各处的计算机通过通信线路连接在一起，构成一个系统，这就是计算机网络。

计算机网络需要一个网络操作系统对整个网络实施管理，并为用户提供统一、方便的网络接口。网络操作系统一般建立在各个主机的本地操作系统基础之上，其功能是实现网络通信、资源共享和保护，提供网络服务和网络接口。

网络操作系统有两种存在方式，一是以独立于本地操作系统的网络操作系统的方式存在，如 Novell 公司的 Netware 局域网操作系统，是一种专门的网络操作系统，二是本地操作系统具有网络功能，如 Windows。

1.3.9　分布式操作系统

计算机网络操作系统能使离散的计算机相互进行通信及资源共享，但它不是一个一体化的系统。在网络操作系统的管理下，如果一个计算机上的用户希望使用网上另一个计算机的资源，则必须指明要访问计算机的地址，而无法在不同计算机上进行分布式协同计算。

大量的实际应用要求一个完整的、一体化的系统。分布式系统中有一个全局分布式操作系统，它负责整个系统的资源分配和调度、任务划分、信息传输、控制协调等工作，并为用户提供一个统一的界面。用户通过这一界面实现所需的操作和使用系统资源，至于操作是在哪台计算机上执行或使用哪台计算机的资源是系统的事，用户不必知道，即系统对用户是透明的。

分布式系统和计算机网络的区别在于操作系统的不同，而不是硬件连接。

1.4　操作系统的特性

本课程以多任务操作系统 Linux 为例讲解操作系统原理，所以在此阐述的是多任务操作系统的特征。各种多任务操作系统分别具有各自的特征，但它们具有如下 4 个共同的基本特征。

1. 并发

在单处理机系统中，并发性是指宏观上有多道程序同时运行，但在微观上是交替执行的。多道程序并发执行能提高资源利用率和系统吞吐量。

多个进程的并发执行由操作系统统一控制，为保证并发进程的顺利运行，操作系统提供了一系列管理机制。

2. 共享

共享是指计算机系统中的资源被多个任务共同使用。共享的理由是：

（1）各用户或任务独占系统资源将导致资源浪费。

（2）多个任务共享一个程序的同一副本，而不是分别向每个用户提供一个副本，可以避免重复开发。

并发和共享是紧密相关的。一方面，资源共享是以进程的并发执行为条件的，若不允许进程的并发执行，就不会有资源的共享；另一方面，进程的并发执行以资源共享为条件，若系统不允许资源共享，程序就无法并发执行。

3. 异步

在多道程序系统中，多进程并发执行，但在微观上，进程是交替执行的，因此进程以“走走停停”的不连续方式运行。由于并发运行环境的复杂性，每个进程在什么时候开始执行，何时暂停，以怎样的速度向前推进，多长时间完成，何时发生中断，都是不可预知的，我们称此种特征为异步性。

4. 虚拟

在操作系统中，虚拟指的是通过某种技术把一个物理实体映射为多个逻辑实体，用户程序使用逻辑实体。逻辑实体是用户感觉上有而实际不存在的事物，例如分时系统中，虽然只有一个 CPU，但在分时系统的管理下，每个终端用户都认为自己独占一台主机。此时，分时操作系统利用分时轮转策略把一台物理上的 CPU 虚拟为多台逻辑上的 CPU，也可以把一台物理 I/O 设备虚拟为多台逻辑上的 I/O 设备，方法是用内存中的输入输出缓冲区来虚拟物理设备，用户程序进行输入输出时，其实是在和缓冲区进行输入输出。

1.5 操作系统的体系结构

操作系统是一个大型软件系统，由许多具有独立功能的程序模块构成，这些程序模块之间必然存在某种关系，即必然按照一定结构组成一个完整的操作系统。一般而言，操作系统有两种结构：层次结构和微内核结构。

1.5.1 层次结构

层次结构操作系统的设计思想是，按照操作系统各模块的功能和相互依存关系，把系统中的模块分为若干层次，任一层（除底层模块）都建立在它下面一层的基础上，每一层仅使用其下层提供的服务。

一个操作系统应划分多少层、各层处于什么位置是层次结构操作系统设计的关键问题，没有固定的模式。一般原则是，接近用户应用的模块在上层，贴近硬件的驱动模块在下层。

处于下层的程序模块往往称为操作系统的内核。这部分程序模块包括中断处理程序、各种设备驱动程序、运行频率较高的模块（如时钟管理程序、进程调度程序、低级通信模块、内存管理模块等）。为提高操作系统的执行效率，操作系统内核一般常驻内存。

1.5.2 微内核结构

微内核结构是 20 世纪 90 年代发展起来的，其基本思想是把操作系统中的基本功能模

块组织为微内核，其他功能模块尽量放到核外，通过调用微内核来实现。微内核结构是对传统内核的提炼，它具有如下优点。

1. 简化内核代码维护工作

大内核的内部关系复杂，牵一发而动全身，如果试图修正错误或扩充功能，设计人员必须十分仔细且要反复测试。相比之下，微内核代码量少，结构简单，容易维护。当增加操作系统功能时，可以只在微内核外进行。

2. 建构灵活

基于微内核可以开发多种类型的操作系统，以满足用户的需要。例如，在微内核上提供一组 UNIX 服务程序，系统对于用户好像是 UNIX 系统。如果提供一组 Windows 服务程序，则好像是 Windows 系统。

3. 安全性高

大多数功能作为服务程序在微内核外运行，与客户程序处于同一地位，因此，当某些服务器出现错误时，通常仅影响自身，不会导致整个系统瘫痪。

4. 方便移植

由于计算机硬件不断更新换代，操作系统的移植成为一个重要问题。与硬件相关的功能全部包含在微内核中，只要修改微内核代码就能适应不同的硬件平台，而不需修改核外的操作系统功能模块，因此使操作系统的移植变得很容易。

微内核技术正成为一种潮流，Windows NT 采用的是 Microsoft 的微内核，美国 Carnegie Mellon 大学开发的 Mach 也是微内核结构的操作系统。

1.6　Linux 介绍

1.6.1　Linux 简介

Linux 系统有两种不同的含义。从技术角度，Linux 指的是由 Linus Torvalds 维护的开放源代码 UNIX 类操作系统的内核。然而，目前大多数人用它表示以 Linux 内核为基础的整个操作系统。从这种意义讲，Linux 指的是包含内核、系统工具、完整的开发环境和应用的类 UNIX 操作系统。

Linux 是 UNIX 操作系统的克隆，可以免费使用，遵循 GPL 声明，可以自由修改和传播。与 Windows 等商业操作系统不同，Linux 完全是一个自由的操作系统。Linux 内核最初是由芬兰籍大学生 Linus Torvalds 和通过 Internet 组织起来的开发小组完成的，其目标是与 POSIX（Portable Operating System Interface，可移植操作系统界面）兼容。Linux 包含了人们希望操作系统拥有的所有功能特性，包括真正的多任务、虚拟内存、世界上最快的 TCP/IP 驱动程序、共享库和多用户支持（这意味着成百上千的人能在同一时刻通过网络、Internet 或连接在计算机串行口上的终端或笔记本电脑/微机使用同一台计算机）。

Linux 现在是个人计算机和工作站上的 UNIX 类操作系统。按照层次结构的观点，在同一种硬件平台上，Linux 可以提供和 UNIX 相同的服务，即相同的用户级和程序员级接

口。Linux 绝不是简化的 UNIX，相反，Linux 是强有力和具有创新意义的 UNIX 操作系统，它不仅继承了 UNIX 的特征，而且在许多方面超过了 UNIX。作为 UNIX 类操作系统，Linux 具有下列基本特征。

（1）是真正的多用户、多任务操作系统。

（2）是符合 POSIX 标准的系统。

（3）提供具有内置安全措施的分层的文件系统。

（4）提供 shell 命令解释程序和编程语言。

（5）提供强大的管理功能，包括远程管理功能。

（6）具有内核的编程接口。

（7）具有图形用户接口。

（8）具有大量有用的实用程序和通信、联网工具。

（9）具有面向屏幕的编辑软件。

大量的高级程序设计语言已移植到 Linux 系统上，因而它是理想的应用软件开发平台，而且，在 Linux 系统下开发的应用程序具有很好的可移植性。同时，Linux 还有许多独到之处：

（1）源代码几乎全部都是开放的。任何人都能通过 Internet 或其他媒体得到 Linux 系统的源代码，并可以修改和重新发布。

（2）可以运行在许多硬件平台上。Linux 系统不仅可以运行在 Intel 系列个人计算机上，还可以运行在 Apple 系列、DEC Alpha 系列、MIPS 和 Motorola 68000 系列上。从 Linux 2.0 开始，不仅支持单处理器的机器，还能支持对称多处理器（SMP）的机器。

（3）不仅可以运行许多自由发布的应用软件，还可以运行许多商品化的应用软件。越来越多的应用程序厂商（如 Oracle、Infomix、Sybase、IBM 等）支持 Linux，而且通过各种仿真软件，Linux 系统还能运行许多其他操作系统的应用软件，如 DOS、Windows、Windows NT 等。

（4）强大的网络功能。Linux 诞生、成长于网络，网络功能相当强大，具有内置的 TCP/IP 协议栈，可以提供 FTP、Telnet、WWW 等服务，同时还可以通过应用程序向其他系统提供服务，如向其他 Windows 用户提供类似于网络邻居的 Samba 文件服务。Linux 系统的另一特征是能充分发挥硬件的功能，因而比其他操作系统的运行效率更高。在个人计算机上使用 Linux，可以将它作为工作站使用。在工作站上使用 Linux，可以使它发挥出更高的性能。Linux 以其特有的性能、功能和可用性必将拥有广泛的应用前景。

1.6.2 UNIX、GNU 与 Linux

Linux 是一种类 UNIX 系统，二者有相当的渊源，同时，Linux 遵循 GNU 的 GPL 许可证，是自由软件家族中的一员，因此，要了解 Linux，就必须先了解他们三者之间的关系。

1. Linux 与 UNIX 系统

Linux 的源头要追溯到最早的 UNIX。1969 年，Bell 实验室的 Ken Thompson 利用一台闲置的 PDP-7 计算机开发了一种多用户多任务操作系统。很快，Dennis Richie 加入了这个

项目，在他们的共同努力下最早的 UNIX 诞生了。Richie 受一个更早的项目 MULTICS 的启发，将此操作系统命名为 UNIX。早期 UNIX 是用汇编语言编写的，第三个版本用一种崭新的编程语言 C 重新设计了。C 是 Richie 设计出来并用于编写操作系统的程序语言。通过这次重新编写，UNIX 得以移植到功能更强的 DEC PDP11/45 与 11/70 计算机上运行。

UNIX 系统正式发表于 1974 年，1975 年的第 6 版引入了多道技术，这时，UNIX 成为真正的多用户分时系统。1980 年，Bell 实验室公布了 VAX11/780 系统平台的 32 位操作系统 UNIX32V。在 UNIX32V 的基础上，UNIX 系统走上了以 AT&T 和加州伯克利分校二者为主的发展道路。1980 年，伯克利先后公布了 UNIX BSD 4.0 和 UNIX BSD 4.1，1983 年公布了 UNIX BSD 4.2，而 AT&T 在 1982、1983 年发布了 UNIX system Ⅲ和 UNIX system Ⅴ。

这样的发展持续了几年之后，人们迫切需要一个统一的 UNIX 系统标准，美国 IEEE 的 POSIX 委员会应运而生，专门从事 UNIX 的标准化工作并按照其定义的标准重新实现 UNIX。在由 AT&T、Sunmicro 公司支持下的 UI（UNIX International）和 DEC、HP、IBM 等公司支持下的 OSF（Open Software Foundation）两大集团的配合下，定义了什么是 UNIX 系统——一个可以运行 UNIX 程序的系统就是 UNIX。也就是说，按照层次结构的观点，只要一个系统能够提供一个 UNIX 的标准界面，包括程序级和用户级的，而不管它内部如何实现，更不管它运行于什么硬件平台，都是一个 UNIX 操作系统。

在此之后，AT&T、加州伯克利分校、UI 及 OSF 先后推出了性能更好、安全性更高的 UNIX 系统。经过多年发展，UNIX 走出实验室，成为了操作系统的主流，几乎每个主要计算机厂商都有自有版本的 UNIX。直到今天，UNIX 系统以其稳定、高效的性能在服务器高端市场中依然占据着绝对优势。

随着 PC 机市场的蓬勃发展，很多公司开发了用于 PC 的 UNIX，如 Microsoft 公司的 Xenix、IBM 公司的 PC UNIX 以及 SCO 公司的 SCO UNIX。但是，这些系统都没有改变微软公司 DOS、Windows 系列产品在 PC 机操作系统的垄断地位。

UNIX 是一个简单却非常优秀的操作系统模型。经过 20 年的发展，更加成熟，对其他操作系统，如 Microsoft（微软）公司的 Windows NT 系列系统及 Linux 系统的发展产生了深远的影响。Linux 系统最初以 UNIX 为原型，以实现 POSIX 标准为目标，自 1991 年诞生，在大量高水平人员组成的开发团队的努力工作下，到 2000 年为止，Linux 核心从 0.01 版发展为 2.4 版。Linux 从 UNIX 的各个流派中不断吸取成功经验，继承 UNIX 的优点，抛弃 UNIX 的缺点，具有稳定高效的处理性能，拥有稳定庞大的用户群体，得到众多厂商有力的支持，成为操作系统发展的热点。基于这个核心构建的 Linux 操作系统被微软公司视为最强有力的竞争对手。从 1998 年到 2000 年的发展形势来看，Linux 在低端服务器市场上已经对 Windows NT/2000 造成了极大的压力。

2. 自由软件运动与 Linux

Linux 只是自由软件家族中的一员，也是最具影响的成员之一。自由软件最早由美国麻省理工学院（MIT）的理查德·斯托尔曼（Richard Stallman）提出，源代码拥有属于全人类的公共知识产权，应该在编写和使用程序的人之间自由传播，而不应作为商人谋求利益的手段。由此可见，自由软件不仅是免费使用的问题，而主要是版权的问题。

在计算机工业发展初期，软件只是硬件的附属品。公司只卖硬件，随系统赠送软件。在这一时期，软件的设计思想、新的算法和软件的源代码在专家、学者和公司研究人员之间自由交流，促进了软件的迅速发展。但是，公司很快认识到软件的价值，对软件实施了版权控制，并限制源代码的发布。许多原本属于公众创造的知识财富成为公司的私有财产。软件的开发方式也发生了变化，变成以公司为主体的封闭的开发模式。UNIX 就是这样的例子。虽然 AT&T 开创了 UNIX 的历史，但 UNIX 的发展与完善却与许多大学的专家、学者的努力分不开。事实上，UNIX 是在协作基础上开发的。当 AT&T 在 20 世纪 70 年代末期对 UNIX 的使用和发布强制实施版权控制后，使早期对 UNIX 发展做出贡献的人们感到沮丧。

自 1984 年起，麻省理工学院（MIT）开始支持 Richard Stallman 的努力，即在软件开发团体中发起支持开发自由软件的运动，这导致了自由软件基金会（Free Software Foundation，FSF）的建立和 GNU 项目的产生。Richard Stallman 的信念是，计算机系统应该对用户开放，软件应该自由使用。在其他人的协作下，他创作了通用公共许可证（General Public License，GPL），对推动自由软件的发展起到了重要的作用。与传统的商业软件许可证不同，GPL 保证任何人拥有共享和修改自由软件的自由，任何人有权取得、修改和重新发布自由软件的源代码，并且规定在不增加附加费用的条件下得到源代码（基本的发布费用除外）。这一规定保证了自由软件总的费用较低，使用 Internet 的情况下是免费的。

GPL 条款规定自由软件的衍生作品必须以 GPL 作为其重新发布的许可证。这一规定保证了自由软件及其衍生作品继续保持自由状态。GPL 条款容许销售自由软件，为公司介入自由软件事业敞开了大门。公司的介入弥补了自由软件的不足，对推动自由软件的应用起了很大作用。自由软件基金会发起人的主要项目是 GNU（GUN 不是 UNIX）。它的目标是建立可自由发布和可移植的类 UNIX 操作系统。开始实施 GNU 项目时，没有多少高质量的自由软件可供使用，所以，为 GNU 项目做出贡献的人们先从系统的应用软件和工具入手。因为 GPL 也是自由软件基金会发表的，所以，GNU 操作系统的许多关键组成部分都置于 GPL 条款的约束之下。GNU 项目本身产生的主要软件包包括 Emacs 编辑软件、gcc 编译软件、bash 命令解释程序和编程语言和 gawk 等，还有许多操作系统必不可少的工具。

除了按 GPL 发布的自由软件之外，还有许多按其他许可证发布的自由软件，如 XWindow 系统、TEX 排版系统和 Perl 语言等。随着时间的推移，GNU 项目将这些软件也包括进来，这些工作为 Linux 操作系统的迅速发展奠定了坚实的基础。

1.6.3 Linux 的历史

Linux 可以说完全是一个互联网时代的产物，是在互联网上产生、发展和不断壮大起来的。

Linux 最初的生成动机应当追溯到 1990 年的秋天。那时的 Linus 正在赫尔辛基大学学习 UNIX 课程，所用教材是 Andrew Tanenbaum 的《操作系统：设计与实施》。因为在学校上机需要长时间排队等待，于是“一气之下，我干脆自己掏钱买了一台 PC”，Linus 回忆说。Linus 在自己的 PC 上，利用 Tanenbaum 教授自行设计的微型 UNIX 操作系统 MINIX 为开

发平台，开发了属于自己的第一个程序。“这个程序包括两个进程，都是向屏幕上写字母，然后用一个定时器来切换这两个进程。”他回忆说，“一个进程写 A，另一个进程写 B，所以我就在屏幕上看到了 AAAA、BBBB 如此循环重复的输出结果。”

Linus 说刚开始的时候他根本没有想到要编写一个操作系统内核。1991 年，他需要一个简单的终端仿真程序来存取 USENET 新闻组的内容，于是，他就在前两个草草编写的进程的基础上又写了一个程序。当然，他把那些 A 和 B 改成了别的东西。“一个进程是从键盘上阅读输入的信息，然后发送给调制解调器，另一个进程则是从调制解调器上阅读发送来的信息，然后送到屏幕上供人阅读”。然而，要实现这两个新的进程，显然还需要一些别的东西，这就是驱动程序。他必须为不同的显示器、键盘和调制解调器编写驱动程序。1991 年的夏天，也就是在他购买了第一台 PC 之后 6 个月，Linus 觉得还需要从网上下载某些文件，为此他必须读写某个磁盘。“于是我又不得不写一个磁盘驱动程序，然后是一个文件系统。一旦你拥有了任务切换器、文件系统和设备驱动程序之后，就拥有了一个 UNIX 或者至少是它的一个内核”。Linux 就以这样一种极其古怪但也极其自然的方式问世了。

这个羽毛未丰的操作系统很可能马上夭折，所以 Linus 没有在 MINIX 新闻组中公布它。他只在赫尔辛基技术大学的一台 FTP 服务器上发了一则消息，说用户可以下载 Linux 的公开版本。“Linux 是我的笔名，但我要是真用它来命名的话，我担心有人会认为我狂妄自大，而不会去认真地对待它。所以我当时选了一个很糟糕的名字：Freax，这个字是由 free（自由）+freak（怪胎）+ x 构成的，幸好，管理这台 FTP 服务器的 Ari Lemmke 根本不喜欢 Freax 这个名字，他最后还是选择了 Linux”。

到 1992 年 1 月止，全世界大约只有 100 多人使用 Linux，但正是他们为 Linux 做了关键性的在线洗礼。他们提供的所有初期上载代码和评论后来被证明对 Linux 的发展至关重要，尤为重要的是那些网上黑客们为了解决 Linux 的错误而编写的许多插入代码段。网上的任何人在任何地方都可以得到 Linux 的基本文件，并可通过电子邮件发表评论或提供修正代码，USENET 还专门为它开辟了一个论坛。于是，Linux 就从最开始的个人思想的产品变成了一副巨大的织锦，成为了由无数志同道合的黑客们发起的一场运动。

1993 年，Linus 的第一个“产品”版 Linux 1.0 是按完全自由发行版权进行发行的。他要求所有源代码必须公开，而且任何人均不得从 Linux 交易中获利。然而半年以后，Linus 开始意识到这种纯粹的自由软件的理想对于 Linux 的发行和发展来说实际上是一种障碍，而不是一股推动力，因为它限制了 Linux 以磁盘复制或 CD-ROM 等媒体形式进行发行的可能，也限制了一些商业公司参与 Linux 的进一步开发并提供技术支持的良好愿望。于是 Linus 决定转向 GPL 版权，这一版权除了规定自由软件的各项许可权之外，还允许用户出售自己的程序备份。

Linux 与 GPL 的结合，使许多软件开发人员相信这是一个有前途的项目，开始参与内核的开发工作，并将 GNU 项目的 C 库、gcc、Emacs、bash 等很快移植到 Linux 内核上。可以说，Linux 项目一开始就和 GNU 项目紧密结合在一起，系统的许多重要组成部分直接来自 GNU 项目。Linux 操作系统的另一些重要组成部分来自加利福尼亚大学 Berkeley 分校的 BSDUNIX 和麻省理工学院的 X Windows 系统项目，这些都是经过长期考验的成果。正

是 Linux 内核与 GNU 项目、BSDUNIX 以及 MIT 的 X11 的结合，才使整个 Linux 操作系统得以很快形成，而且建立在稳固的基础上。

商业软件公司的加盟也使大多数 Linux 的普通用户吃了定心丸。因为在很多人看来，“可自由发行”的软件好像总是和“缺乏技术支持”以及“业余水平”划等号，其实不然，Linux 一开始主要是在一些软件行业高手之间流行，并且很快就在全球范围内集结了一大批职业和业余的技术专家，形成了一个庞大而且热心的支持者群体，他们能够通过网络很快地响应使用者所遇到的任何问题。例如，当 Pentium II 设计上的 Bug 刚一被发现，Linux 就最早地提供了解决方案。1997 年，Linux 支持者群体在众多软件公司中一举胜出，荣获了美国《InfoWorld》杂志的最佳技术支持奖，而这一奖项原本只为商业公司而设立。

当 Linux 走向成熟时，一些人开始建立软件包来简化新用户安装和使用 Linux 的方法，这些软件包称为 Linux 发布或 Linux 发行版本。发行 Linux 不是某个人或某个组织的事，任何人都可以将 Linux 内核和操作系统的其他组成部分组合在一起进行发布。在早期众多的 Linux 发行版本中，最有影响的要数 Slackware 发布，它是当时最容易安装的 Linux 发行版本，在推广应用 Linux 的过程中起了很大作用。Linux 文档项目（LDP）是围绕 Slackware 发布写成的，从出版物的角度看，围绕 Slackware 发布的出版物仍然占主导地位。目前，Red Hat 发行版本的安装更容易，应用软件更多，已成为最流行的 Linux 发行版本，2000 年秋已经发行了 7.0 版本；而 Caldera 则致力于 Linux 的商业应用，它的发展速度也很快。汉化的 Linux 发行版本也有很多，国内自主建立的有 BluePoint Linux、Flag Linux、Xterm Linux 以及美国的 XLinux、TurboLinux 等，这些发行版本大多对安装及使用界面进行了部分汉化（有些采用国际化技术，同时也支持其他语种，在支持多语种方面 XLinux 的工作最引人注目）。

每种发行版本都有各自的优点和弱点，但它们都提供相对完整的应用软件及帮助文档，都使用相同的内核和开发工具，都使用同一个名称——Linux 系统。

1.6.4 Linux 的内核特征

Linux 操作系统的核心稳定而高效，以独占的方式执行最底层任务，保证其他程序的正常运行，是整个系统的核心，具有独特的性质。本节将从操作系统接口、功能及内核结构等几个方面来展示 Linux 核心的特征。

1. 接口特色

按照 POSIX 标准，一个可以运行 UNIX 程序的系统就是 UNIX。Linux 系统提供和一般 UNIX 系统相同的标准界面，包括程序级和用户级的，因此也是一个 UNIX 系统，一般，大家称之为类 UNIX 系统，以区别于其他传统意义上的 UNIX 系统。

在程序级，Linux 系统提供标准的 UNIX 函数库，一个在 Linux 下开发的应用程序，几乎可以不经过任何改动在其他 UNIX 系统下编译执行，完成同样的功能。

Linux 系统向用户同时提供图形和文本用户界面，文本界面是 shell 接口，图形界面是 XWindow 系统。UNIX 下的基本命令在 Linux 下功能和使用方式都完全相同。最早在 UNIX 平台开发的图形用户界面 XWindow 系统在 Linux 系统下运行良好，并可以展示与其他版

本 UNIX 系统上相同甚至更好的效果。更为可喜的是，在 XWindow 系统基础上，自由软件开发者们为 Linux 开发了多种桌面系统，在这样的环境下，用户几乎不再需要传统的文本用户界面，所有操作都可以通过鼠标点击完成。这样的系统有方便快捷的 KDE（K Desktop Environment），基于 CORBA 组件技术，具有图形功能的 GNOME（GNU's Network Object Model Environment）等，它们都遵循 GPL，都处在高速发展阶段，相信它们的功能会更加完善。

桌面系统的发展，基于桌面系统的办公、家用软件的发展，将会使 Linux 操作系统的用户界面更加友好，Linux 系统针对办公用户及普通家庭的普及工作也将具有更明显的竞争力和更美好的前景。

2. 功能特色

Linux 核心最早运行在 Intel 80386 系列 PC 机上，现在，也可以运行在 Apple 系列、DEC Alpha 系列、MIPS 和 Motorola 68000 系列的计算机上，同时，一些改进的嵌入式 Linux 核心还可以运行于手机、家电等设备上。从 Linux 2.0 开始，它不仅支持单处理器的机器，还能支持对称多处理器（SMP）的机器，实现真正的多任务工作。

Linux 系统可以支持多种硬件设备。Linux 系统下的驱动程序开发和 Windows 系统相比要简单得多。最初的硬件设备驱动程序都是由自由软件开发者们提供的，随着 Linux 系统的普及，越来越多的硬件厂商也开始提供设备驱动。

Linux 采用多级分页的存储管理模式，具体的技术特征将在后面介绍。

Linux 自身使用的专用文件系统为 Ext2，可以提供方便有效的文件共享及保护机制，同时，它可以通过虚拟文件系统的技术，支持包括微软系列操作系统所使用的 Fat16、Fat32 和 NTFS 等文件系统在内的几十种现有的文件系统。

Linux 系统具有内置的 TCP/IP 协议栈，可以提供各种高效的网络功能，包括基本的进程间通讯、网络文件服务等。

3. 结构特征

前面曾经介绍过，Linux 内核基本采用模块结构、单内核模式，这使得系统具有很高的运行效率，但系统的可扩展性及可移植性受到一定影响。为了解决这个问题，Linux 使用了附加模块（Modules，也称为模组）技术。利用模块技术，可以方便地在内核中添加新的组件或卸载不再需要的内核组件，而且这种装载和卸载可以动态进行，即在系统运行过程中完成，而不需要重新启动系统。

引入动态的模块技术，可使系统内核具有良好的动态可伸缩性，但是，内核模块的引入也带来了对系统性能、内存利用和系统稳定性的一些影响，可动态装卸的模块需要系统增加额外的资源来记录、管理，而装入的内核模块和其他内核部分一样，具有相同的访问权限，差的内核模块会导致系统不稳定甚至崩溃，特别是一些恶意的内核模块可能对系统安全造成极大危害。

总的来讲，Linux 内核基本采用模块式结构构造，同时加入动态的模块技术，在追求系统整体效率的同时，实现了内核的动态可伸缩性。这样的结构给系统移植带来一定的负面影响，但是，在广大自由软件爱好者们不懈的努力下，Linux 系统仍然不断地推出支持

新硬件平台的新版本，Linux 可以运行的硬件平台超过任何一种商业系统，具有较好的平台适应性。

1.6.5 Linux 的发展及展望

1. 开发模式

自由软件的开发模式不同于以往任何一种软件开发模式。软件工程的发展实现了软件的工程化生产——在经过详细的需求分析之后，进入设计阶段，然后是实现、测试等，整个过程有严格的工作流程、时间限制和质量控制，程序员在整个生产过程中的作用相当于传统工厂里流水线上的工人，只是按照图纸完成某个零部件加工而已，这样的开发模式强调统一规划和集中管理。

自由软件的开发过程完全是另一种情形。一大批广泛分布于世界各地的软件爱好者，以互联网为纽带，通过 BBS、新闻组及电子邮件等现代通讯方式，同时参与一个软件开发项目。一个初步工作的软件雏形首先发布出来，然后大家同时开始工作，分别结合自己的实际经验和需要，寻找软件中的漏洞，提出改进意见，发布在互联网上，很快，有人也发现了漏洞，提出了改进方案，给出补丁，经过这些人分头修整，这个软件好像滚雪球一样，以很快的速度不断完善。在这样的开发模式中，程序员是独立的实体，他们大多用业余时间为自由软件服务，没有工作任务的压力，创作性工作带来的成就感是他们最大的动力。这样的开发模式称为“巴扎”（Bazaar）模式，Bazaar 这个词是英文中的外来词，原文是维吾尔语，原义是街道，引申为市集，我国新疆的很多地区，每月定期举行的市场就称为“巴扎”，大家自发地从各地赶来，带着自己的产品，自由地进行各种交易。在这种开发模式下，软件以平行排错、分头发展的方式快速演进，比传统的模式更能调动程序员的工作热情，软件的开发速度以及软件的质量都有可靠的保证。

自由软件的出现改变了传统的以公司为主体的封闭的软件开发模式，采用了开放和协作的开发模式，无偿提供源代码，允许任何人取得、修改和重新发布自由软件的源代码。这种开发模式激发了世界各地软件开发人员的积极性和创造热情，大量软件开发人员投入到了自由软件的开发中。软件开发人员的集体智慧得到充分发挥，大大减少了不必要的重复劳动，并使自由软件的漏洞得到及时发现和克服。任何一家公司都不可能投入如此强大的人力去开发和检验商品化软件。这种开发模式使自由软件具有强大的生命力。

2. 内核版本

为了确保看似无序的市集开发过程能够有序进行，自由软件一般都必须采取强有力的版本控制措施。

Linux 内核采用的是双树系统，一棵树是稳定树，主要用于发行；另一棵树是非稳定树或者开发树，用于产品开发、改进。一些新特性、实验性改进等首先在开发树中进行。如果在开发树中所做的改进也可以应用于稳定树，那么在开发树中经过测试后，就在稳定树中进行相同的改进。按照 Linus 的观点，一旦开发树经过足够的发展，就会成为新的稳定树，如此周而复始地进行下去。

源代码版本序号的形式为 x.y.z。对于稳定树来说，y 是偶数；对于开发树来说，y 是

比相应稳定树大一的奇数。

这种开发比常规惯例快，因为每一版本所包含的改变比以前少，内核开发人员只需花很短的时间就能完成一个实验开发周期。

当今，Linus 正在率领分布在世界各地的 Linux 内核开发队伍完善他们的作品。Linux 内核 2.x 版本充分显示了 Linux 开发队伍的非凡创造力和市集开发模式的价值。Linux 核心开发者的名单记录在文件/usr/src/linux/CREDITS 中。这些人中的绝大多数都受过高等教育，很多曾从事过博士后工作，工作经历都相当丰富，很多人从事计算机行业，多数人有几年甚至几十年的软件开发经验，毫无疑问，这些人都是精于开发的高手。此外，还有许多志愿参加测试工作并发现系统问题的人，有时他们甚至给出了正确的代码，但这些志愿者没有包括在 CREDITS 中。正是这些人，保证了 Linux 具有稳定可靠的性能。

事实上，UNIX 开始发展时，也采用了类似的开发模式。这使得 UNIX 的安全漏洞比其他操作系统解决得更彻底。从充分发挥开发人员的集体智慧这一点看，采用这种开发模式无疑是一大进步。

3. 国内应用状况

随着 Linux 核心的不断成熟，各种性能稳定、安装方便、支持多语种的发行版本被广泛使用。Linux 得到广大硬件、整机厂商和应用程序厂商的大力支持，这一切，都使得 Linux 这个年轻的系统充满了希望。

由于多种原因，Linux 在国内的推广比国外晚了几年，近年来，更多的软件爱好者开始了 Linux 的学习、应用和研究开发，同时，许多大学还把它作为操作系统课程实验的内容，这些都为 Linux 在中国的推广使用奠定了基础。

Linux 的使用开始于国内的高校和科研单位，最初大家在各地的电子公告牌上研讨问题，随着讨论的深入，他们开始成立各种民间组织，建立自己的主服务器。爱好者们在这些地方下载软件，自由地讨论 Linux 方面的问题，寻找志同道合者切磋，方便而高效地交流信息，为 Linux 的进一步推广和本地化创造了良好的环境。

目前国内较有影响的推广项目是 1997 年 6 月 17 日在国家经济信息中心网上建立的自由软件协会站点，网址是 http://freesoft.cei.gov.cn/freesoft.html，这既是一个大型自由软件库，也是一个自由软件应用的示范项目。整个系统建立在 Linux 基础上，提供 WWW、FTP、DNS、News 和邮件服务。

国内也出现了多家 Linux 发行商，推出多种汉化的 Linux 版本，如 BluePoint、Xterm Linux、Flag Linux 等，同时也提供系统集成、技术支持等服务。

总的来讲，国内 Linux 发展还处于一个比较低的层次，初级入门用户很多，实际应用用户少，而从事自由软件开发的人就更少了。

在国内推广 Linux，让更多的人学习它，熟悉它，在实际工作中应用它，掌握其性能特性和开发技能，在 Linux 平台上实现自己的算法，将有助于 Linux 的推广使用和本地化。

4. 发展方向

Linux 内核本身的发展方向主要是硬件支持、嵌入系统和分布式系统 3 个方面。

提供更多高性能的硬件驱动程序，让更新、更好的硬件迅速在 Linux 系统下工作是 Linux 普及和广泛应用的基础。

随着以计算技术、通信技术为主体的信息技术的快速发展和 Internet 的广泛应用，嵌入式软件成为软件业的新热点。就国内而言，2000 年约有 3 亿多台彩电、4000 多台 VCD、2000 多万台学习机、7000 多万部寻呼机和 4000 万部手持电话、掌上 PC 以及数千万辆汽车，它们都是潜在的信息电器和嵌入式系统的应用平台。同时，近年来 Internet 发展迅速，据权威机构统计，1998 年中国 Internet 用户超过 200 万，1999 年 6 月超过 400 万，2000 年已经超过 800 万。面对如此巨大的电子产品市场和潜在用户群，嵌入式软件的应用前景十分广阔，而 Linux 系统本身的开放特性以及稳定的性能都比较适合作为开发嵌入系统的原型，国内外都有这样的研究项目，也有相当成功的案例。

分布式系统是当前操作系统发展的另一个重要领域。以 Linux 内核为基础，按照自由软件开发模式，发展高性能的自由分布操作系统是操作系统发展的必然趋势。

此外，Linux 上的桌面系统、应用软件，尤其是软件开发工具是 Linux 发展的重要方面。桌面系统直接关系到 Linux 界面的友好性与易用性。应用软件关系到系统的可用性，而在自由软件开发模式中引入软件工程新技术和成功经验有助于快速开发 Linux 平台上的应用软件。

本章小结

操作系统是一个大型软件系统，它的基本功能是资源管理和方便用户使用计算机系统（提供用户接口）。资源管理功能包括处理机管理、存储器管理、设备管理、文件管理；用户接口包括作业级接口、程序级接口。

操作系统是裸机之上的第一层软件，通过控制和管理计算机硬件资源，为应用软件提供运行环境。

在诸多因素的推动下，操作系统开始形成并不断发展。在计算机发展初期，人们手工操作计算机，存在着严重的人机矛盾，计算机运行效率很低。为了解决人机矛盾，引入了批处理技术，在监督程序的控制下，自动转换作业。单道程序顺序执行技术的资源利用率低，系统吞吐量小，因此引入了多道程序并发执行技术，与批处理技术结合构成多道批处理系统。多道批处理系统充分提高了资源利用率和系统吞吐量，但用户使用不方便，因此开发了既能保证资源利用率，又能提供人机交互的分时系统。在对计算机实时处理和高可靠性有专门要求的计算机应用领域中引入了实时操作系统。微机的出现导致了微机操作系统的产生，计算机互联的需求导致了网络操作系统的产生。分布式系统对分散的计算机资源进行统一管理，极大地满足了用户的需求，提高了系统的可靠性。

操作系统的基本特征是并发、共享、异步、虚拟。

操作系统的结构一般有两种类型：层次结构、微内核结构。微内核结构是操作系统结构的发展方向。

习　题　一

1-1　什么是操作系统？它有哪些基本功能和基本特征？
1-2　操作系统发展的动力是什么？
1-3　批处理系统的目标是什么？
1-4　为什么要引入多道程序并发执行技术？
1-5　试分析单道与多道批处理系统的优缺点。
1-6　为什么要引入分时操作系统？
1-7　分时系统是怎样实现的？
1-8　实时操作系统应用在哪些场合？
1-9　实时系统有哪些特点？
1-10　操作系统的结构有哪些类型？分别阐述。
1-11　处理机管理有哪些主要功能？它们的主要任务是什么？
1-12　内存管理有哪些主要功能？它们的主要任务是什么？
1-13　设备管理有哪些主要功能？它们的主要任务是什么？
1-14　文件管理有哪些主要功能？它们的主要任务是什么？
1-15　操作系统提供了哪些人机接口？

第 2 章　进程管理

本章学习目标

进程是操作系统中一个非常重要的概念。程序的运行是通过进程来完成的。在层次结构的操作系统中，进程不仅是系统分配资源的基本单位，而且是 CPU 调度的基本单位。进程管理是操作系统最重要的功能之一。本章主要介绍进程的概念、状态、构成以及 Linux 进程的相关知识。通过本章的学习，读者应该掌握以下内容：

- 进程的概念
- 进程的描述、状态及转换
- 进程的特征
- Linux 进程的描述及进程通信
- 进程的同步与互斥
- 线程的概念及特征

2.1　进程的基本概念

2.1.1　程序的顺序执行及其特征

1. 程序的顺序执行

程序是人们要计算机完成的一些指令序列，是一个按严格次序、顺序执行的操作序列，是一个静态的概念。我们把一个具有独立功能的程序独占处理机，直到最后结束的过程称为程序的顺序执行。

例 1：有一个程序，要求先输入数据，再做相应的计算，最后输出结果并用打印机打印。分别用 I、C、P 代表以上 3 个程序段，这样，上述 3 个程序段的执行顺序为：I→C→P。

对于一个程序段中的各个语句来说，也有一个执行顺序问题，如在 C 程序段中有下列 3 条语句：

```
S1：x:=a+b;
S2：y:=x*5;
S3：z:=x+y;
```

则程序段 C 的执行过程是：S1→S2→S3。

2. 程序顺序执行时的特征

（1）顺序性。处理机的操作严格按照程序所规定的顺序执行，即只有前一个程序段

完成才执行下一个程序段，上一条指令完成再去执行下一条指令。

（2）封闭性。程序是在封闭环境下运行的。程序运行时独占全机资源，资源的状态除初始状态外，只有该程序本身才能改变它。程序执行的最终结果由给定的初始条件决定，不受外界因素的影响。

（3）可再现性。顺序执行的最终结果可再现，也就是说它与执行速度及执行的时刻无关，只要输入的初始条件相同，无论何时重复执行该程序，结果都是相同的。

2.1.2　程序的并发执行及其特征

1. 并发执行的概念

程序的并发性是指多道程序在同一时间间隔内同时发生，这是为了增强计算机系统的处理能力，提高资源的利用率所采取的一种同时操作技术。

例 2：在例 1 的程序中，3 个程序段存在着 I→C→P 前趋关系。现在考虑 3 个功能类似于例 1 的程序：程序 1、程序 2、程序 3，它们所执行的程序段及前趋关系分别为：

程序 1：I1→C1→P1

程序 2：I2→C2→P2

程序 3：I3→C3→P3

在某一时刻，系统的状态可能是打印机在为程序 1 打印输出结果、CPU 在为程序 2 进行运算，同时输入设备正在忙于输入程序 3 的数据，即 3 个程序段 P1、C2、I3 同时执行。

程序的并发执行可总结为一组在逻辑上互相独立的程序或程序段在执行过程中，其执行时间在客观上互相重叠，即一个程序段的执行尚未结束，另一个程序段的执行已经开始的一种执行方式。

2. 程序并发执行时的特征

程序的并发执行虽然提高了系统吞吐量，但也产生了下述一些与顺序执行时不同的新特征。

（1）间断性。程序在并发执行时，由于它们共享系统资源，以及为完成同一项任务而相互合作，致使这些并发执行的程序之间形成了相互制约的关系。例如，例 2 中并发执行的 3 个程序，当程序 3 正在执行程序段 P3，而此时程序段 C2 已执行完成时，应当使用打印机执行 P2，而此时打印机被程序 3 占据不能使用，致使程序 2 暂停等待打印机；又如，I1 输入完成要进行计算，而此时 CPU 正在运行 C2，则此时程序 1 就暂时不能继续运行，需等待 CPU。相互制约将导致并发程序具有“执行—暂停—执行”这种间断性的活动规律。

（2）失去封闭性。程序在并发执行时，是多个程序共享系统中的各种资源，因而这些资源的状态将由多个程序来改变，致使程序的运行失去了封闭性。这样，某程序在执行时，必然受到其他程序的影响，例如，当处理机被某个程序占有时，另一程序必须等待。

（3）不可再现性。程序在并发执行时，由于失去了封闭性，也将导致其失去可再现性，例如，有两个循环程序 A 和 B，它们共享一个变量 N，初值为 0。

程序 A：

```
{repeat
```

```
    N=N+1;
  until false
  }
```

程序 B：

```
  {repeat
    print(N);
  until false;
  }
```

程序 A 和程序 B 以不同的速度运行，由于程序 A 和程序 B 速度的差异，出现的结果可能不同。例如，当程序 A 和程序 B 运行的速度相近时，打印的结果为 1、2、3、…；而当程序 A 的执行速度是程序 B 的 2 倍时，打印的结果可能是 2、4、6…。

程序在并发执行时，由于失去了封闭性，其计算结果与并发程序的执行速度有关，从而使程序失去了可再现性，程序经过多次执行后，虽然执行时的环境和初始条件相同，但得到的结果却各不相同。

从上面的讨论看，由于程序的顺序性、间断性和不可再现性，用程序段作为描述其执行过程和共享资源的基本单位，既增加了操作系统设计和实现的复杂性，又无法反映操作系统应具有的程序段执行的并发性、用户随机性，以及资源共享等特征。也就是说，用程序作为描述其执行过程以及共享资源的基本单位是不合适的。这就需要一个既能描述程序的执行过程，又能用来共享资源的基本单位，这个基本单位被称为进程。

2.1.3 进程的定义与特征

1. 进程的定义

进程是操作系统中最基本、最重要的概念之一。进程的概念是 20 世纪 60 年代初期，首先在 MIT（美国麻省理工学院）研制的 Multics 系统和 IBM 的 TSS/360 系统中引用的。从那时起，人们对进程下过许多定义。现列举其中的几种。

（1）进程是程序的一次执行。

（2）进程是可以和别的进程并发执行的计算。

（3）进程就是一个程序在给定活动空间和初始条件下，在一个处理机上的执行过程。

（4）进程是程序在一个数据集合上的运行过程，是系统进行资源分配和调度的一个独立单位。

（5）进程是动态的、有生命周期的活动。内核可以创建一个进程，最终将由内核终止该进程使其消亡。

下面讨论进程和程序之间的关系。例如，在 Linux 系统下使用编辑器 vi 进行编辑，同时打开多个窗口，编辑多个不同名称的文件，vi 编辑器是一个可执行程序，不同的文件就是不同的操作数据，而对应于这些文件同时打开的每一个编辑窗口都对应着一个进程，每一个进程都处于不同状态。

如果说程序是提供计算机操作的一组工作流程的话，进程就是具体的工作过程，按照同样的工作流程，针对不同的原料，可以同时开始多个工作过程，得到多种不同的成品。

这种工作流程和工作过程的关系就可以类比为程序和进程的关系。

进程和程序是两个完全不同的概念，但又有密切联系，它们之间的主要区别如下。

（1）程序是静态的概念，本身可以作为一种软件资源长期保存，而进程是程序的一次执行过程，是动态的概念。

（2）进程是一个能独立运行的单位，能与其他进程并发执行。进程是作为资源申请和调度单位存在的；通常的程序不能作为一个独立运行的单位而并发执行。

（3）程序和进程不存在一一对应的关系。一个程序可由多个进程共用，一个进程在其活动中又可顺序地执行若干个程序。一个程序运行一次，便创建了一个进程；那么同一个程序运行 10 次，就产生了 10 个进程。

（4）各个进程在并发执行过程中会产生相互制约的关系，造成各自前进速度的不可预测性，而程序本身是静态的，不存在这种异步特征。

2. 进程的特征

从进程与程序的区别可以看出，进程具有如下特征。

（1）动态性。进程是进程实体的执行过程，因此，动态性是进程最基本的特性。进程由创建而产生，由调度而执行，因得不到资源而暂停执行，并因撤消而消亡，可见，进程有一定的生命周期。

（2）并发性。这是指多个进程实体，共存于内存中，能在一段时间段内同时执行。并发性是进程的重要特征，同时也是操作系统的重要特征。提高并发性，可以提高系统的效率。

（3）独立性。进程是一个能独立运行的基本单位，同时也是系统中独立获得资源和独立调度的基本单位。凡未建立进程的程序，都不能作为一个独立的单位参加运行。

（4）异步性。这是指进程按各自独立的、不可预知的速度向前推进，或者说，进程按异步方式运行。这一特征将导致程序执行的不可再现性，因此，在 OS 中必须采取某种措施来保证各程序之间能协调运行。

（5）结构特征。从结构上看，进程实体是由程序段、数据段及进程控制块 3 部分组成，也称这 3 部分为进程映像。

2.1.4　进程的基本状态及转换

1. 进程的 3 个基本状态

根据进程活动的动态性及异步性，一个进程从创建到撤消，要经过不同的状态。我们可以从系统的资源角度出发进行讨论，系统资源可分为 CPU、外部事件（包括资源、时钟中断等）。在操作系统中，进程通常有如下 3 种基本状态。

（1）就绪状态（Ready）。进程的外部条件满足，但因为其他进程已占用 CPU，所以暂时不能运行。

（2）执行状态（Running）。外部条件满足，进程已获得 CPU，其程序正在执行。在单处理机系统中，只有一个进程处于执行状态。

（3）阻塞状态（Blocked）。进程因等待某种事件发生（如等待其他进程释放资源、等

待一个时钟中断、等待其他进程发来的信号等），而暂时不能运行的状态，称为阻塞状态，也称为等待状态。也就是说，处于阻塞状态的进程尚不具备运行条件，即使 CPU 空闲，也无法使用。系统中处于这种状态的进程可能有多个，通常将它们排成一个队列，也有的系统根据阻塞原因的不同将这些进程排成多个队列。

2. 进程状态的转换

进程并非固定处于某个状态，它将随着自身的推进和外界条件的变化而发生变化。对于一个系统，处于就绪状态的进程，在调度程序为之分配了处理机后，该进程便可执行，相应地，它由就绪状态转变为执行状态。正在执行的进程也称为当前进程，如果因分配给它的时间片已用完而被暂停执行时，该进程便由执行状态回到就绪状态；一个处在执行状态的进程，如果因发生某事件而使进程的执行受阻，使之无法继续执行，该进程将由执行状态转变为阻塞状态；一个处于阻塞状态的进程，当它所需的外部事件满足，它应由阻塞状态变为就绪状态。如图 2-1 所示给出了进程的 3 种基本状态及各状态之间的转换关系。

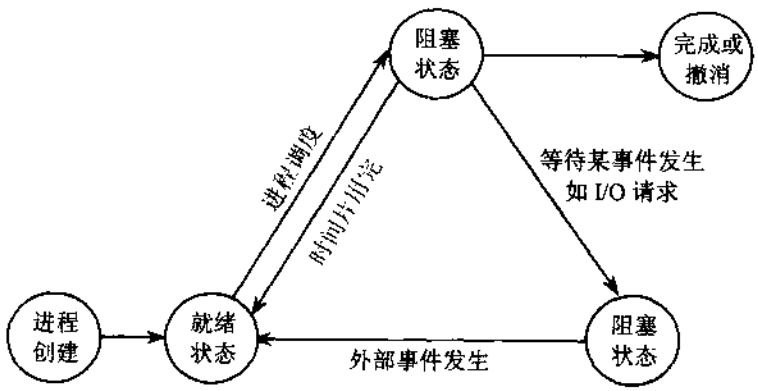


图 2-1 进程的基本状态及转换

3. 引入挂起状态时的进程状态

除了上述 3 种基本状态以外，很多系统中又引入了挂起状态。所谓挂起状态，实际上就是一种静止状态。一个进程被挂起后，不管它是否在就绪状态，系统都不分配给它处理机。

（1）引入挂起状态的原因。

1）终端用户的请求。当终端用户在自己的程序运行期间发现问题时，希望进程静止下来，暂停运行。

2）父进程请求。有时父进程希望挂起自己的某个子进程，以便考察和修改该子进程，或者协调各子进程之间的活动。

3）系统负荷的需要。当实时系统中的工作负荷较重，系统的实时任务受到影响时，将一些不重要的进程挂起。

4）操作系统的需要。操作系统有时希望挂起某些进程，以便检查运行中的资源使用情况或进行统计。

（2）进程状态的转换。

以进程的基本状态为基础，引入挂起状态以后，进程的状态可分为运行、活动就绪、活动阻塞、静止就绪、静止阻塞状态。

一个进程从活动状态被挂起，系统调用 Suspend 原语。被挂起的进程要想重新进入到活动状态，则必须调用 Active 激活原语进行激活（即转换为活动状态），然后才有可能执行。

因此，引入挂起状态后，进程之间的状态转换除了 4 种基本状态转换以外，又增加了以下几种。

1）活动就绪—静止就绪。当进程处于活动就绪状态时，使用 Suspend 将该进程挂起后，该进程便转变为静止就绪状态。处在静止就绪状态的进程不能被进程调度程序选中。

2）活动阻塞—静止阻塞。当进程处于活动阻塞状态时，调用 Suspend 原语将该进程转变为静止阻塞状态。当该进程所等待的事件发生以后，该进程将从静止阻塞状态进入静止就绪状态。

3）静止就绪—活动就绪。处于静止就绪状态的进程若被激活原语 Active 激活后，就进入活动就绪状态。

4）静止阻塞—活动阻塞。处于静止阻塞状态的进程若用激活原语 Active 激活后，将变换为活动阻塞状态。

如图 2-2 所示给出了具有挂起状态的进程状态转换关系。

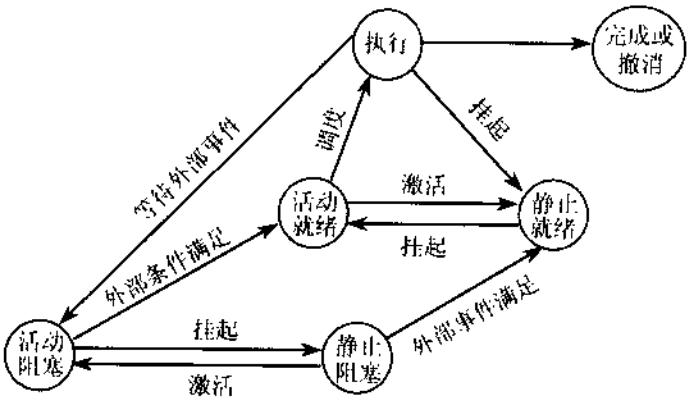


图 2-2　具有挂起状态的进程状态转换

2.1.5　Linux 进程的状态

Linux 系统的一个任务总体上有以下几种状态。

（1）运行状态（Running）。Linux 系统中进程的运行状态表示该任务是“活的”，即存在的。实际包含了上述基本状态中的执行和就绪两种状态，进程实际是正在运行还是处于就绪状态准备运行，要靠当前是否占有 CPU 资源来区分。该状态对应 state 取值为 TASK_RUNNING。

（2）等待状态（Waiting）。进程正在等待某个外部事件。Linux 系统把基本的等待状态进一步细化为可中断的等待状态和不可中断的等待状态两种。处于等待状态的进程都在

等待某个事件或某个资源，可中断等待状态的进程可以被信号唤醒而进入就绪状态等待调度，而不可中断等待状态的进程是因为硬件资源无法满足，不能被信号唤醒，必须等到所等待的资源得到之后由特定的方式唤醒。

（3）中断处理状态（Interrupt Routine）。当硬件发出一个中断信号时（可能是用户键入了一个字符，或时钟每隔 10ms 发出的一次中断等），中断处理例程被激活。此状态对应 state 取值为 TASK_RUNNING。

（4）系统调用期间（System Call）。系统调用通过软件中断启动。一个系统调用可能将相应的进程挂起，等待一个事件发生。此状态对应 state 取值为 TASK_RUNNING。若此状态包含等待状态，则还对应取值 TASK_INTERRUPTIBLE、TASK_UNINTERRUPTIBLE、TASK_STOPPED、TASK_ZOMBIE 等。

（5）从系统调用返回（Return From System Call）。每个系统调用后以及每个慢中断处理完毕后都自动进入该状态。在该状态下，要查看是否需要调用调度程序，是否有信号要处理。调度程序可能将该进程切换为就绪状态，同时激活另一进程。

（6）就绪状态（Ready）。处于此状态的进程正在竞争处理机，但此刻处理机正在为另一个进程服务。此状态对应 state 取值为 TASK_RUNNING。

如图 2-3 所示为 Linux 系统进程的状态转换情况。采取一定的简化措施，按照进程是否占有处理机为依据，把进程的运行状态分为执行和就绪两种状态，等待状态统一考虑，不再区分是否可中断。

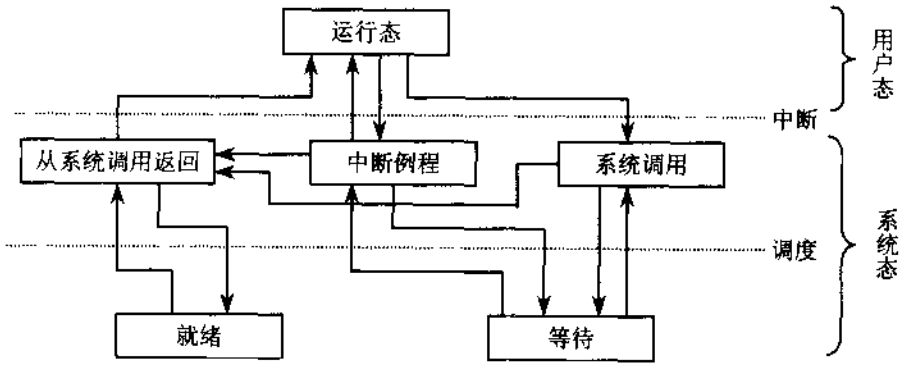


图 2-3 Linux 任务状态转换图

上述进程状态原则上应与 state 中的取值一一对应，但在具体实现时，state 取值与上述的实质状态划分并不一一对应。Linux 系统内核在进程控制块中用 state 成员描述进程当前的状态，并明确定义了 5 种进程状态。

（1）TASK-RUNNING 状态，Linux 系统中的运行状态实际包含了上述基本状态中的执行和就绪两种状态。进程是正在运行还是处于就绪状态准备运行，要靠当前是否占有 CPU 资源来区分。因为有 current 变量，所以两类状态的进程不会发生混淆。

（2）TASK-INTERRUPTIBLE 状态，可中断的等待状态。进程正在等待某些事件。Linux 系统把基本的等待状态进一步细化为可中断的等待状态和不可中断的等待状态两种。处于

这种状态的进程都在等待某个事件或某个资源，可中断等待状态的进程可以被信号唤醒而进入就绪状态等待调度。

（3）TASK-UNINTERRUPTIBLE 状态，不可中断等待状态。不可中断等待状态的进程是因为硬件资源无法满足，不能被信号唤醒，必须等到所等待的资源得到之后由特定的方式唤醒。

（4）TASK-ZOMBIE 状态，僵死状态。由于某些原因进程被终止，这个进程所占有的资源全部释放之后，还保存着 PCB 信息，这种占有 PCB 但已被撤消的进程就处于僵死状态。

（5）TASK-STOPPED 状态，暂停状态。处于暂停状态的进程一般都是由运行状态转换而来，等待某种特殊处理。如处于调试跟踪的程序，每执行到一个断点，就转入暂停状态，等待新的输入信号。

2.2　进程的描述

进程是一个动态的概念，描述程序的一次运行活动。进程存在于系统的内存中，是操作系统可感知、可控制的动态实体，是系统分配各种资源、进行调度的基本单位。那么，从处理机的活动角度来看，如何描述进程呢？进程的活动是通过在 CPU 上执行一系列程序和对相应数据进行操作来体现的，因此程序和它的操作数据是进程存在的实体，但这两者仅是静态的文本，没有反映出其动态性，为此，还需要一个数据结构来描述进程当前的状态、本身的特性等，这种数据结构称为进程控制块（Process Control Block，简称进程控制块）。

进程的程序部分描述进程所要完成的功能，而数据结构集是执行时必不可少的工作区和操作对象，这两部分是进程完成所需功能的物质基础。由于进程的这两部分内容与控制进程的执行及完成进程功能直接相关，因而，在大部分多道操作系统中，这两部分内容放在外存中，直到该进程执行时再调入内存。所以进程实体通常是由程序、数据集合和 PCB 这 3 部分构成，也称为“进程映像”。下面分别介绍进程的 PCB 结构、程序与数据结构集。

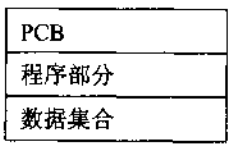


图 2-4　进程的组成模型

如图 2-4 所示为进程的一般组成模型。

2.2.1　进程控制块 PCB

如上所述，PCB 包含一个进程的描述信息、控制信息和资源信息，有些系统中还有进程调度等待所使用的现场保护区。PCB 集中反映一个进程的动态特征。进程并发执行时，由于资源共享，使得各进程之间相互制约，显然，为了反映这些制约关系和资源共享关系，在创建一个进程时，应首先创建其 PCB，然后根据 PCB 中的信息对进程实施有效的管理和控制。当一个进程完成其功能后，系统释放 PCB，进程也随之消亡。

一般来说，PCB 记录了进程的全部控制信息，庞大而复杂。根据操作系统的要求不同，进程的 PCB 所包含的内容也多少有所不同，可以按照功能大概分成 4 个组成部分：进程标

识符、处理机状态、进程调度信息、进程控制信息。

1. 进程标识符

进程标识符用于惟一地标识一个进程。一个进程通常有以下两种标识符。

（1）进程内部标识符。在所有的操作系统中，为每一个进程赋予一个惟一的数字标识符，它通常是一个进程的序号。设置内部标识符主要是为了方便系统使用。

（2）进程外部标识符。它由创建者提供，通常是由字母、数字组成，往往由用户（进程）在访问该进程时使用。为了描述进程的家族关系，还应设置父进程标识及子进程标识。此外，还可设置用户标识，用以表示拥有该进程的用户。

2. 处理机状态

处理机状态信息主要由处理机的各种寄存器中的内容组成。处理机在运行时，许多信息都放在寄存器中。当处理机被中断时，这些信息都必须保存在 PCB 中，以便在该进程重新执行时能从断点继续执行。处理机的寄存器包括通用寄存器、指令计数器、程序状态字 PSW、用户栈指针。

PCB 中设有专门的 CPU 现场保护结构，以存储退出执行时的进程现场数据。

3. 进程调度信息

PCB 中还存放一些与进程调度和进程对换有关的信息。

（1）进程状态。指明进程当前的状态，作为进程调度和对换的依据。

（2）进程优先级。用于描述进程使用处理机的优先级别的整数，优先级高的进程应优先获得 CPU。

（3）进程调度所需要的其他信息。它们与所采用的进程调度算法有关，如进程已等待 CPU 的时间、进程已运行的时间等。

（4）事件或阻塞原因。指进程由执行状态转变为阻塞状态所等待发生的事件。

4. 进程控制信息

进程控制信息包括：

（1）程序和数据的地址，指进程的程序和数据所在的内存或外存地址，以便再调度到该进程执行时，能从 PCB 中找到其程序和数据。

（2）进程同步和通信机制，指实现进程同步和进程通信时必需的机制，如消息队列指针、信号量等，它们可能全部或部分地放在 PCB 中。

（3）资源清单，是一张列出了除 CPU 以外、进程所需的全部资源及已经分配到该进程的资源清单。

（4）链接指针，给出了本进程 PCB 所在队列的下一个进程 PCB 的首地址。

当系统创建一个新进程时，就为它建立一个 PCB；当进程终止后，系统回收其 PCB，该进程在系统中就不存在了。所以，PCB 是进程存在的惟一标志。

2.2.2 进程控制块的组织方式

系统中有许多进程。处于就绪状态和阻塞状态的进程可能有多个，而阻塞的原因又可以各不相同。为了对所有进程进行有效的管理，常将各进程的 PCB 用适当的方式组织起来。

一般各进程的 PCB 的组织方式有线性方式、链接方式和索引方式。

1. 线性方式

线性方式最简单、最容易实现。操作系统预先确定整个系统中同时存在的进程的最大数目和静态分配空间，把所有进程的 PCB 都放在这个表中，如图 2-5 所示。这种方式存在的主要问题是限定了系统中同时存在的进程的最大数目。当很多用户同时上机时，就会造成无法为用户创建新进程的情况。更严重的缺点是，在执行 CPU 调度时，为选择合理的进程投入运行，经常要对整个表进行扫描，降低了调度效率。



图 2-5　PCB 线性队列示意图

2. 链接方式

链接方式是经常采用的方式，其原理是按照进程的不同状态分别将其放在不同的队列，如图 2-6 所示。在单 CPU 情况下，处于运行状态的进程只有一个，可以用一个指针指向它的 PCB。处于就绪状态的进程可以是若干个，它们排成一个或多个队列，通过 PCB 结构内部的链指针把同一队列的 PCB 链接起来。该队列的第一个 PCB 由就绪队列指针指向，最后一个 PCB 的链接指针为 0，表示链结束。CPU 调度程序把第一个 PCB 由该队列中摘下（设仅一个队列），令其投入运行。新加入就绪队列的 PCB 按照某一调度算法插入。阻塞队列可以有多个，各对应不同的阻塞原因。当某个等待条件得到满足时，则可以把对应阻塞队列上的 PCB 送到就绪队列中，正在运行的进程如出现缺少某些资源而未能满足的情况，就变为阻塞状态，加入相应阻塞队列。Linux 操作系统就是应用这种进程控制块组织方式。

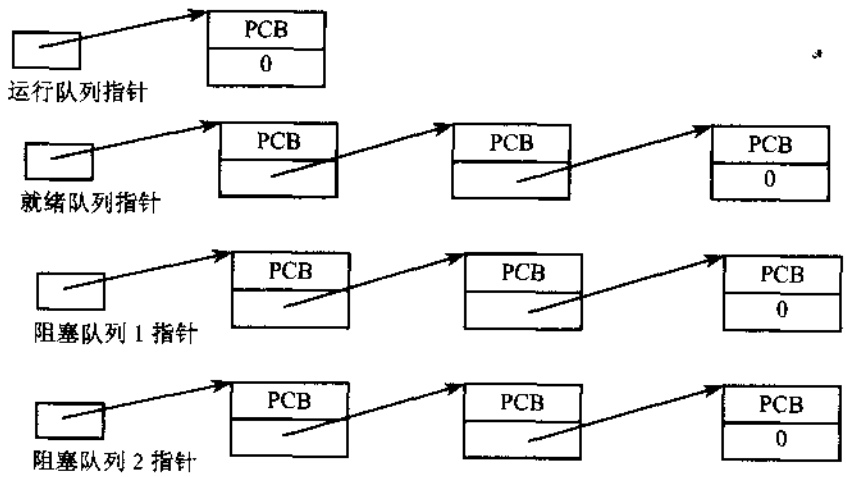


图 2-6　PCB 链接队列示意图

3. 索引方式

系统根据所有进程的状态建立几张索引表，如就绪索引表、阻塞索引表等，并把各索

引表在内存的首地址记录在内存的一些专用单元中。在每个索引表的表目中，记录具有相应状态的某个 PCB 在 PCB 表中的地址，如图 2-7 所示。

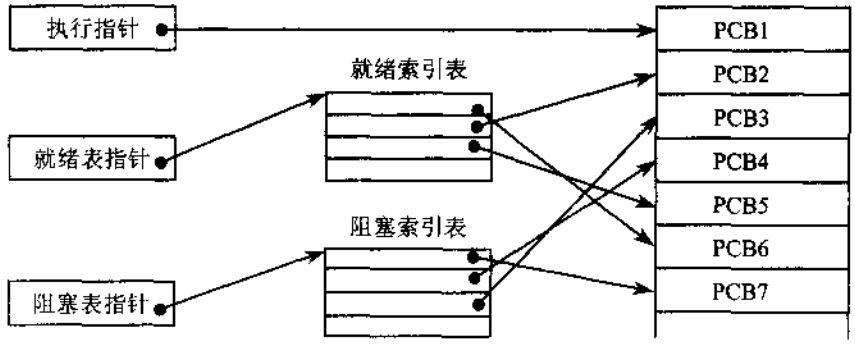


图 2-7 PCB 索引结构示意图

2.2.3 Linux 进程的 PCB

Linux 系统中的进程称为任务。该系统的进程控制块 PCB 用一个称为 task-struct 的结构体来描述，具体定义参看 Linux 源代码文件 include/linux/sched.c。Linux 系统 PCB 包含以下信息。

1. 进程描述信息

通过进程描述信息，Linux 系统可以惟一地确定某一个进程的基本情况，可以了解该进程所属的用户及用户组等信息，同时还能确定这个进程与所有其他进程之间的关系。这些描述信息包括进程号、用户和组标识以及描述进程家族关系的连接信息。

（1）进程标识号（pid，Process Identifier）。进程一旦创建，就由系统给定一个进程序号，叫做进程标识号。这是一个不小于 0 的整数，它与系统中的进程是一一对应的。以后系统对该进程的操作，都是通过这个进程标识号来实现的。

（2）用户和组标识（User and Group Identifier）。Linux 系统中有不同的用户和组标识，主要用来控制进程对系统文件的访问权限，实现系统资源的安全访问。Linux 系统中所有的文件都有所有者和允许的权限，这些权限描述了系统使用者对文件和目录的使用权。基本的权限是读、写和可执行，这些权限被分配给文件的所有者、同组用户和系统中的其他用户三类用户。

（3）连接信息（Links）。用来记录进程的家族关系，如父进程标识号等。

2. 进程控制信息

（1）进程当前状态。

（2）调度信息。记录进程的优先数等。

（3）记时信息。

（4）通信信息。Linux 支持典型的 UNIX 进程间通信机制——信号、管道，也支持 System V 通信机制——共享内存、信号量和消息队列。

3. 进程资源信息

进程资源信息记录了与该进程有关的存储器的各种地址和资料、文件系统以及打开文件的信息等。

4. CPU 现场信息

task-struct 是 Linux 系统的进程控制块（PCB），通过对 PCB 的操作，系统为进程分配资源并进行调度，最终完成进程的创建和撤销。

2.3　进程控制

进程和处理机管理的一个重要任务是进程控制。所谓进程控制，就是系统使用一个引起具有特定功能的程序段来创建、撤消进程以及完成进程各状态间的转换，从而达到多进程高效率并发执行和协调、实现资源共享的目的。通常，把系统态下执行的某些具有特定功能的程序段称为原语。原语的特点是不可被中断。原语可分为两类，一类是机器指令级的，其特点是执行期间不允许中断，是一个不可分割的基本单位；另一类是功能级的，其特点是作为原语的程序段不允许并发执行。这两类原语都在系统态下执行，都是为了完成某个系统管理所需要的功能而被高层软件所调用。

系统在创建、撤消一个进程以及要改变进程的状态时，都要调用相应的程序段来完成这些功能。用于进程控制的原语有创建原语、撤消原语、阻塞原语和唤醒原语等。

2.3.1　进程的创建与终止

1. 进程的创建

在多道程序环境下，只有进程才能在系统中运行，因此，为使程序能够运行，必须为它创建进程。导致进程创建的事件有用户登录、作业调度、为用户提供服务等。

一旦系统发现要求创建进程的事件后，便立刻调用创建原语 Creat()，通过下述步骤创建一个进程。

（1）申请空白 PCB。为新进程申请获得惟一的进程标识符，并从 PCB 集合中索取一个空白 PCB。

（2）为新进程分配资源。包括新创建进程的程序、数据及用户栈所需的内存空间。此时，系统必须知道新进程所需内存的大小。对于批处理作业，其大小可在用户提出创建进程要求时提供。如果为应用进程创建子进程，也应在该进程提出创建进程的请求中给出所需内存的大小。对于交互型作业，用户可以不给出内存要求而由系统分配一定的空间。

（3）初始化进程控制块。PCB 的初始化包括初始化标识信息，如进程标识符、父进程标识符；处理机状态信息，使程序计数器指向程序的入口地址，使栈指针指向栈顶；处理机控制信息，将新建进程的状态设置为就绪状态（活动就绪或静止就绪状态）；进程的优先级等。

（4）将新建进程插入就绪态队列。

2. 进程的终止过程

如果系统中发生了要求进程终止的事件后，操作系统便调用进程终止原语，终止该进程。

（1）根据被终止进程的标识符，从 PCB 队列中检索出该进程的 PCB，从中读出该进程的状态。

（2）若被终止进程正处于执行状态，应立即终止该进程的执行，该进程被终止后应重新进程调度。

（3）检查该进程有无子孙进程，若有，应将其所有子孙进程终止。

（4）释放终止的进程所占有的资源，将其归还它的父进程或系统。

（5）将被终止的进程从它的 PCB 队列中移出。

3. 进程阻塞与进程唤醒

通过进程的创建原语可以创建一个进程；撤消原语完成的任务是在进程运行完成或被强行撤消后将进程消亡，但这还不能完成进程各种状态的转换，例如，进程从运行状态到阻塞状态，或从阻塞状态转换为就绪状态，这需要通过进程之间的同步或通信机构来实现，也可直接使用阻塞原语和唤醒原语来实现。

（1）进程阻塞。当一个进程所等待的某一事件尚未发生时，该进程调用阻塞原语 block() 将自己阻塞，并转换为等待状态。阻塞原语的功能是将调用阻塞原语的进程的 CPU 现场送到该进程的现场保护区，置该进程的状态为等待，并将该进程插入到相应的等待队列中，然后转进程调度程序，选择一个就绪状态的进程使之执行。

（2）进程的唤醒。进程由运行状态转变为等待状态，是由于进程必须等待某一事件的发生，所以处于等待状态的进程不可能将自己唤醒，只有当该进程所等待的外部事件发生时，才由发生该事件的进程调用唤醒原语 wakeup()将它唤醒。

唤醒原语的执行过程是，首先把被阻塞进程从等待该事件的阻塞队列中移出，将其 PCB 中的现行状态由阻塞状态改为就绪状态，然后，将它的 PCB 插入到就绪态队列。

2.3.2 几个相关的 Linux 系统调用

在 Linux 系统中，系统向用户提供了一些对进程进行控制的系统调用。

1. fork()系统调用

Linux 利用 fork()系统调用创建一个新进程。fork()系统调用的格式是：

```
int fork();
```

通常情况下，设返回值为 int pid，调用格式为：

```
pid=fork();
```

其中，返回值 pid 意义如下：

pid=0：创建子进程成功，表示从子进程返回。

pid>0：创建子进程成功，表示从父进程返回，pid 的值为新创建的子进程标识号。

pid=-1：创建失败。

fork()函数是通过复制来创建子进程的，子进程继承父进程的上下文，是父进程的一

个副本，与父进程使用同一段代码。在该系统调用之后，两个代码相同的进程并发执行。

2. exec()系统调用

fork()系统调用创建的子进程和父进程执行的是同一段代码，但实际上，它们完成不同的工作。在 Linux 系统中，当由 fork()系统调用创建一个子进程后，可再利用 exec()系统调用执行另一个程序。

exec()系统调用有两种基本调用格式，下面给予说明。

格式一：
```
int execl(path,arg0[,arg1,…,argn],0)
    char *path,*arg 0,*arg1,…arg n;
```
格式二：
```
int execv (path,arg v)
    char *path,*argv[ ];
```

3. exit()系统调用

对于一般的用户进程，在其任务完成后应被尽快撤消。Linux 系统用 exit()系统调用来实现进程的自我终止。通常，父进程在创建子进程时，应在进程的末尾写一条 exit()，使子进程自我终止。该系统调用的格式说明如下：

```
void exit(status)
    int status;
```

其中，status 是返回给父进程的一个整数，如 0、1 等。

4. wait 系统调用

该系统调用将调用进程挂起，直至其子进程因暂停或终止而发来软中断信号为止。wait 系统调用的格式说明如下：

```
int wait(stat_loc)
int stat_loc;
```

其中，stat_loc 是用户空间的一个地址，它含有子进程的退出状态码。

2.3.3　进程的阻塞与唤醒

进程的创建原语和撤消原语完成了进程从无到有、从存在到消亡的过程。被创建后的进程最初处于就绪状态，然后经调度程序选中后进入执行状态。实现进程从执行状态到等待状态，又由等待状态到就绪状态转换的两种原语分别为阻塞原语和唤醒原语。

阻塞原语在一个进程期待某一事件发生，但发生条件还不满足时，被该进程自己调用来阻塞自己。阻塞原语在阻塞一个进程时，由于该进程正处在执行状态，故应先中断处理机并保存该进程的 CPU 现场，然后将被阻塞进程插入阻塞队列中，再转到进程调度程序，选择新的就绪进程投入运行。阻塞原语的实现流程如图 2-8 所示。

当等待队列中的进程所等待的事件发生时，等待该事件的所有进程都将被唤醒。唤醒一个进程有两种方法，一种是由系统进程唤醒，另一种是由事件发生进程唤醒。当系统进程唤醒等待进程时，系统进程将该事件发生通知等待进程。因此，该进程便进入就绪队列。等待进程也可由事件发生进程唤醒，这时，事件发生进程与被唤醒进程之间是合作关系。因此，唤醒原语既可被系统进程调用，也可被事件发生进程调用。调用唤醒原语的进程被称为唤醒进程。唤醒原语首先将被唤醒进程从相应的等待队列中摘下，将其置为就绪状态，

送入就绪队列。之后，唤醒原语既可返回原调用程序，也可转向进程调度。唤醒原语的实现过程如图 2-9 所示。

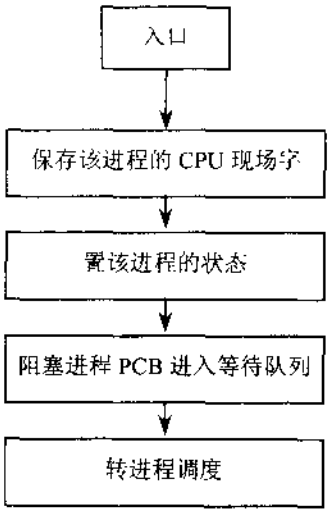


图 2-8 阻塞原语的实现

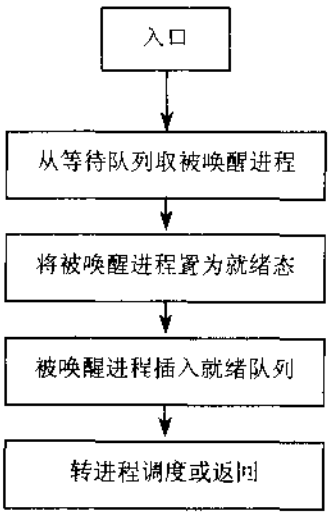


图 2-9 唤醒原语的实现

2.4 进程的同步与互斥

在操作系统中引入进程及进程并发性的概念后，增加了系统的效率，同时，由于资源有限导致了进程之间的资源竞争和共享，因此产生了一些问题。例如，当多个进程同时申请一台打印机时，如果不加限制，很可能使多个进程的输出结果交织在一起，产生错误。本节主要介绍并发执行的进程使用临界资源时，同步与互斥的关系及控制方法。

2.4.1 临界资源的概念

1. 临界资源

两个或两个以上的进程不能同时使用的资源称为临界资源（Critical Resource，简称 CR）。临界资源可能是一些独占设备，如打印机、磁带机等；也可能是一些共享变量、表格、链表等。

例 1：假设一个非常简单的飞机订票系统有两个终端 T1 和 T2，执行下面的并发进程。

```
cobegin                    /*表示并发执行的进程*/
  var n:integer;           /*系统中剩余票的数量，若 n>0 则可卖票*/
repeat T1(n);              /*重复执行的进程 T1*/
repeat T2(n);              /*重复执行的进程 T2*/
coend
```

并发进程 T1 和 T2 的执行序列如下：

（1）进程 T1 的定义

```
进程：T1()
begin
     read(n);
          if n>=1 then
     {卖一张票；
     n:=n-1;
     write(n);  }
end;
```

（2）进程 T2 的定义

```
进程：T2()
begin
  read(n);
       if n>=1 then
  {卖一张票；
  n:=n-1;
  write(n);  }
end;
```

考虑此时进程 T1 和进程 T2 是并发执行的，若剩最后一张票，即 n=1 时，T1 进程可卖一张票。假设 T1 正在执行“卖一张票”语句时，进程 T2 刚好执行到 if 语句，因为这时 n=1，故判断通过，因此，进程 T2 也可卖一张票，这样，最后一张票卖给了两个顾客，从而产生了错误。错误的原因就是对于共享变量 n，两个进程没能有限制地使用。如果在进程 T1 和 T2 的程序段中定义共享变量 n 为临界资源，则可避免该错误的发生。

2. 临界区

不论硬件临界资源，还是软件临界资源，多个进程必须互斥地对其进行访问。每个进程中访问临界资源的那段代码称为临界区（Critical Section）。所以，若能保证各进程互斥地进入临界区，便可实现各进程对临界资源的互斥访问。因此，每个进程在进入临界区以前，应先对欲访问的临界资源进行检查，看它是否正在被访问。如果此临界资源正在被访问，则该进程不能进入临界区；若此刻该临界资源未被使用，则该进程可进入该临界区对该资源进行访问，并设置正在被访问标志。因此，必须在临界区前面增加一段用于进行上述检查的代码，把这段代码称为进入区；相应地，在临界区后面再加一段用于检查退出临界区的代码，称为退出区，用于将临界区正被访问的标志恢复为未被访问的标志。进程中除去上述进入区和退出区外，其他部分的代码称为剩余区。这样，可将一个访问临界资源的进程描述如下：

```
repeat
   进入区；
   临界区；
   退出区；
   剩余区；
until false;
```

2.4.2　进程的互斥与同步

1. 同步与互斥的概念

进程互斥是指多个进程不能同时使用同一个临界资源 CR，即两个或两个以上进程必须互斥地使用临界资源，或不能同时进入临界区 CS。这是由各进程共享某些资源引起的，如系统只有一台打印机，两个进程都要使用。为了保证打印结果的正确和方便使用，只能一个进程用完打印机后，另一个进程才能使用。两个进程对打印机的申请没有顺序关系，谁都可能先提出请求。为了解决这个问题，使用前，各进程先申请打印机，如打印机空闲，则分配给它，并做上占用标记，以后一直占用，直到本次用完后，释放打印机，清除标记，这时另一个进程的申请才能得到满足。因此，这两个逻辑上完全独立、毫无关系的进程，

由于竞争同一个资源而相互制约，就称为进程的互斥。

进程同步是指有协作关系的进程之间不断地调整它们之间的相对速度或执行过程，以保证临界资源的合理利用和进程的顺利执行。实现进程同步的机制称为进程同步机制。不同的同步机制实现同步的方法也可能不同，但一般都借助一个中间媒体来实现，如信号量操作、加锁操作等。

在计算机中存在这种关系的进程很多，如两个进程合作使用同一个缓冲区。设进程 A 负责往缓冲区中输入数据，进程 B 负责从同一缓冲区中输出数据。当进程 A 将数据输满缓冲区，则只有当进程 B 将该数据读出后，进程 A 才能继续使用该缓冲区，否则将造成数据丢失。此时，进程 A 和进程 B 之间就形成了同步关系。

2. 同步机制应遵循的规则

为实现进程互斥地进入自己的临界区，可用软件方法，比较多的是在系统中设置专门的同步机构来协调各进程间的运行。所有同步机构都应遵循下列准则。

（1）空闲让进。并发进程中某个进程不在临界区时，不阻止其他进程进入临界区。

（2）忙则等待。并发进程中的若干个进程申请进入临界区时，只允许一个进程进入。当已有进程进入临界区时，其他申请进入临界区的进程必须等待，以保证对临界资源的互斥访问。

（3）有限等待。访问临界资源的进程应保证在有限时间内进入自己的临界区，避免因长时间申请临界资源得不到满足，而一直继续等待，陷入“等死”状态。

（4）让权等待。当进程不能进入自己的临界区时，应立即释放处理机，以免进程陷入“忙等”状态。

2.4.3 锁机制

实现互斥的一种软件是采用锁机制，即提供一对上锁（Lock）和开锁（UnLock）原语，以及一个锁变量 w 或者是锁位（1 个 bit）。

当进程进入临界区之前，首先测试 w 的状态。w＝1，表示上锁，该资源已被占用；w＝0 表示开锁，该资源已空闲，未被占用。加锁及解锁原语可描述如下。

加锁原语：

```
Lock w:
L: if w=1 then goto L
  Else w:=1
```

开锁原语：

```
UnLock w:
  w: =0;
```

这种利用锁机制解决互斥问题的方法安全可靠，但是效率不高，因为上锁原语中 w＝1 的情况要反复循环测试，直到时间片耗完为止，因此浪费了处理机的时间。

2.4.4 信号量机制

1965 年，荷兰的 Dijkstra 提出了一种广义锁机制或称为计数锁的同步机制，既能解决互斥，又能解决同步，是一种非常有效的同步工具，后来被加以改造，并通过长期而广泛

的应用，使信号量同步机制有了很大的发展，进而发展为信号量集同步机制。申请和释放临界资源的两个原语操作为 wait 操作和 signal 操作，有时也称为 P 操作和 V 操作。wait(S)和 signal(S)有时也称为 P 原语操作和 V 原语操作，P 和 V 分别是荷兰语 Passeren 和 Verhoog 的第一个字母，相当于英文中 Pass 和 Increment 的意思。

信号量（Semaphore）也叫做信号灯，是在信号量同步机制中用于实现进程的同步和互斥的有效数据结构。我们可以为每类资源设置一个信号量。信号量有多种类型的数据结构，如整型信号量、记录型信号量、AND 型信号量及信号量集等。

1. 整型信号量

它是信号量的最简单的类型，也是各种信号量类型中必须包含的类型。整型信号量的数值表示当前系统中可用的该类临界资源的数量。如设置整型信号量 s，则 s 值的意义为：

s>0，表示系统中空闲的该类临界资源的个数。

s=0，表示系统中该类临界资源刚好全部被占用，而且没有进程在等待该临界资源。

s<0，s 的绝对值表示系统中等待该类临界资源的进程的个数。

wait(S)和 Signal(S)操作可描述为：

```
wait(S): while S≤0 该进程等待;
         S:=S-1;
Signal(S): S:=S+1;
```

2. 记录型信号量

记录型信号量的数据结构由两部分构成。例如，定义记录型信号量 S，有如下描述：

```
type semaphore=record
s:integer;
L:list of process;
End
  S: semaphore;
```

即 s 的值表示系统中可用的该类临界资源的数量，而 L 为进程链表指针，指向等待该类资源的 PCB 队列。

下面以记录型信号量为例进行说明。

设变量 S 为如上定义的记录型信号量，则 wait(S)操作和 signal(S)操作的流程如图 2-10 和 2-11 所示。

申请临界资源的原语 wait()操作可描述为：

```
procedure wait(S)
  var S: semaphore;
  begin
   s: =s-1;
if s≥0 then 本进程继续;
else
将本进程放入阻塞态队列;
转进程调度
 end
```

释放临界资源的原语 signal()操作可描述为：

```
procedure signal(S)
  var S: semaphore;
  begin
  s: =s+1;
  if s≤0 then 唤醒指针 L 所指的阻塞态进程;
  end
```

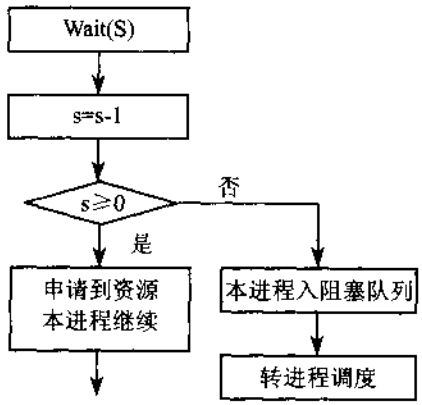


图 2-10 Wait 操作原语流

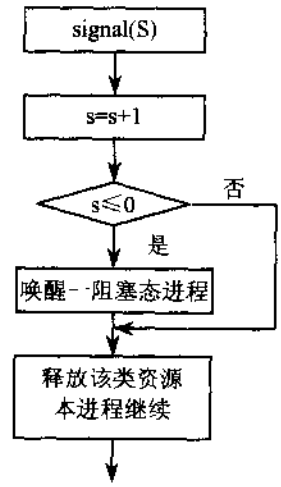


图 2-11 signal 操作原语流

在记录型信号量机制中，s 的初值表示系统中该类资源的可用数目，因而又称为资源信号量，每次对它的 wait 操作，即申请该类一个单位的临界资源，描述为 s:=s-1；当 s≥0 条件满足时，表示在没有做减 1 操作之前 s≥1，因此，本进程可继续执行；当 s≥0 条件不满足时，表示在减 1 操作之前 s<1，系统中没有空闲的该类资源，因此进程应调用 block 原语，将该进程的 PCB 插入由指针 L 指向的阻塞队列。因此，该机制遵循了“让权等待”准则，s 的绝对值表示在该信号量链表中已阻塞进程的数目。对信号量的每次 signal 操作表示执行进程释放一个单位的该类临界资源，因此操作为 s:=s+1，表示资源数目加 1。若加 1 后仍是 s≤0，则表示在做加 1 操作之前 s≤-1，说明在该信号量链表中，仍有因等待该资源被阻塞的进程，所以还应调用 wakeup 原语，将指针 L 所指链表中的第一个等待进程唤醒；若 s≤0 不成立，则表示系统中没有等待该类资源的进程，因此本进程只需释放它所占用的该类资源，继续执行。

信号量 S 除初始化外，仅能通过两个标准的原语操作 wait(S)和 signal(S)来访问。因为 wait(S)和 signal(S)是两个原语操作，因此，它们在执行时是不可中断的。

3. AND 型信号量

AND 同步机制的基本思想是将进程在整个运行过程中需要的所有资源，一次性全部分配给进程，待进程使用完成后再一起释放。只要有一个资源尚未分配给进程，其他所有可能分配的资源也不能分配给它。

AND 型信号量集机制可描述如下：

```
Swait(S1, S2, …, Sn)
  if S1≥1 and … and Sn≥1 then
     for i: =1 to n do
     Si: =Si-1;
     endfor
  else
  将该进程放入阻塞态队列;
  endif
Ssignal(S1, S2, …, Sn)
   for i: =1 to n do
   Si=Si+1;
  唤醒所有因 Si 不满足而进入阻塞队列的进程;
endfor;
```

4. 信号量集

如果某进程一次需要 N 个某类资源时，就要进行 N 次 wait 操作，这使系统的效率较低，又可能造成死锁。

信号量集机制的基本思想是在 AND 型信号量集的基础上进行扩充，进程对信号量 S_i 的测试值为 t_i（用于信号量的判断，即 $S_i \geq t_i$ 表示资源数量低于 t_i 时不予分配），占用值为 d_i（用于信号量的增减，即 $S_i=S_i-d_i$ 和 $S_i=S_i+d_i$）

```
Swait(S1, t1, d1; …; Sn, tn, dn);
Ssignal(S1, d1; …; Sn, dn);
```

一般“信号量集”的几种特定情况如下：

（1）Swait(S, d, d)表示每次申请 d 个资源，当少于 d 个时，便不分配。

（2）Swait(S, 1, 1)表示互斥信号量。

（3）Swait(S, 1, 0)作为一个可控开关（当 S≥1 时允许多个进程进入临界区；当 S=0 时禁止任何进程进入临界区）。

（4）“信号量集”未必成对使用 Swait()和 Ssignal()，如一起申请，但可以不一起释放。

5. 利用信号量实现进程的同步和互斥

考虑例 1 中所描述的出现错误的情况，可以用 wait(s)和 signal(s)操作解决这个问题，避免错误的发生。对临界资源 n 设一互斥信号量 S，将原进程 T1 及 T2 做如下修改：

```
进程：T1()
   begin
   wait(s);
   read(n);
   if n>=1 then
   {  n:=n-1;
      write(n);
   signal(s);
   卖一张票；  }
   end;
```

```
进程：T2()
  begin
    wait(s);
    read(n);
    if n>=1 then
    { n:=n-1;
    write(n);
    signal(s);
    卖一张票；  }
end;
```

2.5 进程同步问题举例

2.5.1 生产者—消费者问题

1. 问题的描述

生产者—消费者（Producer-Consumer）问题是一个著名的进程同步问题。它是这样描述的：有一群生产者进程在生产产品，并将这些产品提供给消费者进程去消费，为方便生产者进程与消费者进程能并发执行，在两者之间设置了一个具有 n 个缓冲区的缓冲池，生产者进程将它所生产的产品放入一个缓冲区中；消费者进程可从一个缓冲区中取走产品去消费。显然，应规定消费者进程不能到一个空缓冲区中去取产品；生产者进程不能将产品放入一个已装满产品且尚未被取走的缓冲区中。

2. 问题的分析

用一个数组表示上述具有 n（0，1，…，n-1）个缓冲区的缓冲池。设一个输入指针 in，表示下一个可存放产品的缓冲区，每当生产者进程生产一个产品并放入该缓冲区后，in:=in+1；设一个输出指针 out，表示下一个可从中取得产品的缓冲区，每当消费者进程从此取走一个产品后，out:=out+1。考虑此处的缓冲池构成了循环缓冲，故当输入或输出指针加 1 时表示为 in:=(in+1)mod n；out:=(out+1)mod n.。当(in+1)mod n=out 时，表示缓冲池满；当 in=out 时则表示缓冲池空。用整型变量 counter 表示该缓冲池中满缓冲区的个数，显然，每当生产者进程向缓冲池中存放一个产品后，counter 加 1；每当消费者进程从缓冲区中取走一件产品后，counter 减 1，如图 2-12 所示。

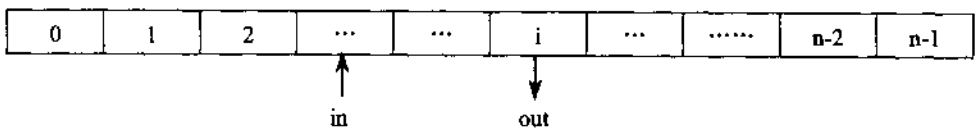


图 2-12 生产者—消费者问题

假设初始情况下缓冲池为空，即 counter＝ 0 ，为在生产者–消费者问题中实现各进程的同步，可设下列信号量（假设初始情况下没有进程使用缓冲池，且缓冲池中各缓冲区都是空的）：

mutex：互斥使用缓冲池信号量，由于初始情况下无进程使用缓冲池，故初值 mutex=1。

empty：使用缓冲池中空缓冲区的信号量，由于初始情况下所有缓冲区为空，故初值 empty=n。

full：使用缓冲池中满缓冲区的信号量，由于初始情况下没有缓冲区存放产品，故初值 full＝ 0 。

设开始时生产者进程存放产品和消费者进程取走产品，都从第 0 号缓冲区开始，并设这些生产者和消费者地位相当，只要缓冲池未满，生产者便可将消息送入缓冲池，只要缓冲池未空，消费者便可从缓冲池中取走一个消息。

3. 算法及程序

通过以上分析，对生产者—消费者问题描述如下：

```
Var mutex, empty, full:semaphore:=1,n,0;        /*定义信号量并赋初值*/
  buffer:array[0, …, n-1] of item;
  in, out: integer:=0, 0;                  /*定义存取指针的初始位置*/
  begin
  parbegin
```

生产者进程：

```
proceduer:
begin
       repeat
        …
      生产一件产品;
        …
       wait(empty);
       wait(mutex);
       将产品放入下一个缓冲区;
       in:=(in+1) mod n;
       signal(mutex);
       signal(full);
       until false;
     end
```

消费者进程：

```
consumer:
begin
       repeat
       wait(full);
       wait(mutex);
       从下一个缓冲区中取走一件产品;
       out:=(out+1) mod n;
       signal(mutex);
       signal(empty);
       消费这件产品;
       until false;
     end
   parend
  end
```

4. 生产者—消费者问题中的注意事项

（1）在每个程序中用于实现互斥的 wait(mutex)和 signal(mutex)必须成对出现，即对临界资源使用前必须申请，使用完成后必须做释放操作。

（2）对资源信号量 empty 和 full 的 wait 和 signal 操作也需要成对出现，但它们分别处于不同的进程中，以保证生产者进程和消费者进程的同步及交替执行（若 n=1，则生产者进程和消费者进程只能严格交替执行；若 n>1，则在消费者进程不执行的前提下，生产者进程最多可重复执行 n 次；反之，消费者进程也同样）。

（3）特别注意，在每个进程中，多个 wait 操作顺序不能颠倒，而 signal 操作的次序无关紧要。临界资源 mutex 的使用是非常重要的，请读者考虑一下为什么。若将计算进程比喻为生产者进程，将打印进程比喻为消费者进程，则 wait(empty)在计算进程中，而 signal(empty)在打印进程中，计算进程若因执行 wait(empty)而阻塞，以后将由打印进程将它唤醒。由于生产者—消费者问题是相互合作的进程关系的一种抽象，把并发进程的同步和互斥问题一般化，就可以得到一个抽象的一般模型，即生产者和消费者问题。把系统中使用同一类资源的进程称为该资源的消费者，而把释放同类资源的进程称为该资源的生产者。例如，在输入时，输入进程是生产者，计算进程是消费者；而在输出时，计算进程是生产者，打印进程是消费者，因此，该问题有很大代表性及实用价值。

2.5.2 读者—写者问题

1. 问题的提出

文件 F 可以被多个并发进程共享，将访问该文件的进程按访问方式分为两类，一类只能读共享对象的内容，把这类进程称为读进程或读者；另一类进程要更新（写）共享对象文件 F，把这些进程称为写进程或写者。试用 Wait、Signal 操作解决各进程间的同步问题。

2. 问题的分析

显然，多个读者同时读一个共享对象是可以的，然而一个写者不能与其他任何读者或写者同时共享该文件，即使用共享文件时，一个写进程与其他所有进程都是互斥的，但多个读进程之间不存在互斥现象，如图 2-13 所示。

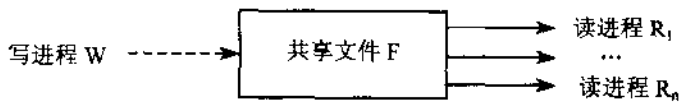


图 2-13 读者—写者问题

设读进程为 reader，写进程为 writer。为实现 reader 与 writer 进程间的同步与互斥，设如下变量及信号量：

wmutex：互斥使用该共享文件信号量，如写进程 write 与读进程 reader 在使用文件时是互斥的；共享文件只有一个，设初始情况未被使用，则初值为 1。

readcount：整型变量，表示正在读的进程个数，初值为 0。

rmutex：计数器 readcount 的信号量。因为 readcount 是一个可被多个 reader 进程访问的临界资源，为此设一信号量。设初始状态下无进程读和写，故 rmutex 的初值设为 1。

由于多个进程可以同时读，因此只要有一个 reader 进程在读，其他 reader 进程便不必申请该共享文件，直接读即可；若无文件在读，则第一个读文件的进程必须做申请该文件的操作。只要有 read 进程在执行，则不允许 writer 进程执行。因此，仅当 readcount=0，即无 reader 进程在读时，reader 进程才需要执行 wait(wmutex)操作。若 wait(wmutex)操作成功，reader 进程便可读，相应地，做 readcount+1 操作。同理，仅当 reader 进程在执行了 readcount 减 1 操作后其值为 0 时，才需执行 signal(wmutex)操作，以便让 writer 进程写。

3. 算法及程序

读者—写者问题可描述如下：

```
Var:rmutex, wmutex:semaphore:=1,1;
  Readcount:integer:=0;
  begin
  parbegin
```

读者进程：

```
Reader:
begin
  repeat
     wait(rmutex);
     if readcount=0 then wait(wmutex);
     readcount:=Readcount+1;
     signal(rmutex);
      …
     进行读操作;
      …
     wait(rmutex);
     readcount: =readcount-1;
     if readcount=0 then signal(wmutex);
     signal(rmutex);
   until false;
end
```

写者进程：

```
      writer:
      begin
          repeat
            wait(wmutex);
            执行写操作;
            signal(wmutex);
          until false;
       end
   parend
end
```

4. 注意事项及提示

（1）对于写进程，共享文件是临界资源；对于读进程，该文件不是临界资源。

（2）整型变量 readcount 是临界资源，所以使用前后要进行 Wait、Signal 操作。

2.5.3　哲学家进餐问题

1. 问题的提出

设有 5 个哲学家围坐在一张圆桌前吃饭。桌上有 5 支筷子，在每人之间放一支，如图 2-14 所示。哲学家要吃饭时，只有分别从左、右两边都拿到筷子时，才能吃饭。如果筷子已在他人手上，则该哲学家必须等待到他人吃完后才能拿到筷子；任何一个哲学家在自己

未拿到两支筷子吃饭之前，决不放下自己手里的筷子。试描述 5 位哲学家吃饭的进程。

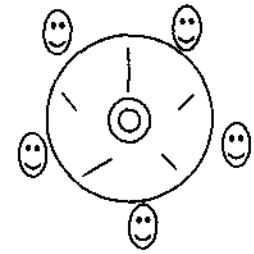


图 2-14 哲学家就餐问题

2. 问题分析

放在桌子上的筷子是临界资源，在一段时间内只允许一位哲学家使用。为了实现对筷子的互斥使用，可以为每一支筷子设置一个信号量，由这 5 个信号量构成信号量数组：

```
Var chopstick: array [0, …, 4] of semaphore;
```

设初始条件下，所有哲学家都未吃，故所有信号量均被初始化为 1。

3. 实现方法

假设每一位哲学家拿筷子的方法都是先拿起左边的筷子，再拿起右边的筷子，则第 i 位哲学家的活动可描述为：

```
P.()
begin
Var chopstick: array [0, …, 4] of semaphore=[1,1,1,1,1];
repeat
          wait(chopstick [i] );
          wait(chopstick [(i+1) mod 5] );
          eat;
          ..
          signal(chopstick [i] );
          signal(chopstick [(i+1) mod 5] );
          think;
  until false;
end
```

这种算法存在一个问题。假设 5 个哲学家同时拿起左边的筷子，那么再去拿右边的筷子时就会产生死锁。

4. 不产生死锁的哲学家就餐问题算法

（1）至多允许有 4 位哲学家同时去拿左边的筷子，最终能保证至少有一位哲学家进餐，并在用毕时能释放出他用过的两支筷子，从而使更多的哲学家能够进餐。

（2）仅当哲学家的左、右两支筷子均可用时，才允许他拿起筷子进餐。

（3）规定奇数号哲学家先拿他左边的筷子，然后再去拿右边的筷子，而偶数号哲学家则相反，最后总会有一位哲学家能获得两支筷子。

2.6 进程通信

进程间的信息交换称为进程通信。上一节介绍的进程互斥与同步就是一种进程间的通信方式。操作系统可以被看作是由各种进程组成的，如用户进程、系统进程、计算进程、打印进程等，这些进程都具有各自的独立功能，且大多数由于外部需要而产生并执行。通常，进程间的通信分为控制信息的传送与大量信息的传送两种。有时，将进程间控制信息

的交换称为低级通信，而把进程之间大批量数据的交换称为高级通信。由于进程的互斥与同步交换的信息量较少且效率较低，因此称这种通信方式为低级通信方式，相应地，也称 wait 和 signal 操作为低级的通信原语。低级通信通常传送一个或几个字节的信息，以达到控制进程执行速度的作用。高级通信要传送大量的信息，而不是为了控制进程的速度，因此，仅通过 P、V 操作或锁的方法无法实现进程的高级通信。

高级通信方式可分为共享存储器系统、消息传递系统和管道通信系统。在共享存储器系统中，相互通信的进程共享某些数据结构或共享存储区；在进程之间的消息传递系统中，进程间的数据交换以消息为单位，用户直接利用系统提供的一组通信原语来实现通信，消息传递系统可分为消息缓冲通信和信箱通信；管道是用于连接读进程和写进程以实现它们之间通信的共享文件，向管道提供输入的发送进程以字符流形式将大量数据送入管道，而接受管道输出的接收进程可从管道中接收数据。

2.6.1　共享存储器系统

1. 共享存储器系统的类型

在共享存储器系统中，相互通信的进程共享某些数据结构或共享存储区，进程之间能通过这些空间进行通信。通常，根据进程之间共享对象的类型将共享存储器系统通信分为如下两种类型。

（1）基于共享数据结构的通信方式。在这种通信方式中，要使各进程共用某些数据结构，以实现各进程间的信息交换。如在生产者—消费者问题中，就是用有界缓冲区这种数据结构来实现通信的。在这里，公用数据结构的设置及对进程间同步的处理，都是程序员的职责，这无疑增加了程序员的负担，而操作系统只需提供共享存储器，因此，这种通信方式效率低，只适用于传通相对少量的数据。

（2）基于共享存储区的通信方式。为了进行进程间的高级通信，在存储器中划出一块共享存储区，各进程可通过对共享存储区中的数据的读或写来实现通信。进程在通信前，先向系统申请获得共享存储区中的一个分区，并指定该分区的关键字；若系统已经给其他进程分配了这样的分区，则将该分区的描述符返回给申请者，然后，由申请者把获得的共享存储区连接到本进程上；此后，便可像读、写普通存储器一样访问该共用存储分区。

2. Linux 共享存储区通信的实现

（1）共享存储区的建立。当进程要利用共享存储区与另一进程进行通信时，必须先利用系统调用 shmget()建立一块共享存储区，并提供该共享存储区的名字 key 和共享存储区以字节为单位的长度 size 等参数。若系统中已经建立了指定的共享存储区，则该系统调用将返回该共享存储区的描述符 shmid；若尚未建立，便为进程建立一个指定大小的共享存储区。

（2）共享存储区的操纵。如同消息机制一样，可以用系统调用 shmctl()对共享存储区的状态信息进行查询，如长度、所连接的进程数、创建者标识符等；也可设置或修改其属性，如共享存储区的许可权、当前连接的进程计数等；还可用来对共享存储区加锁或解锁，以及修改共享存储区标识符等。

3. 共享存储区的附接与断开

当进程已经建立了共享存储区或已获得了其描述符后，还应利用系统调用 shmat()将该共享存储区附接到用户给定的某个进程的虚地址 shmaddr 上，并指定该存储区的访问属性，即指明该区是只读，还是可读可写。此后，该共享存储区便成为该进程虚地址空间的一部分。进程可采取与对其他虚地址空间一样的存取方法来访问。当进程不再需要该共享存储区时，再利用系统调用 shmdt()把该区与进程断开。

4. 几个相关的系统调用

Linux 系统利用几个相关的系统调用实现共享存储区通信。以下列出了几个常用的系统调用及格式。

（1）shmget(key,size,flag)。

该函数使用头文件如下：

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
```

语法格式：

```
shmid=shmget (key, size, flag)
```

该系统调用返回值为整型。

功能：获得一个共享存储区，若成功，其返回值为该共享存储区的描述符。

参数定义：

```
key-t key;
int size, flag;
```

其中，size 是存储区的字节数；key 是该共享存储区的名字；flag 是用户设置的标志或访问权限。flag 可取下列值：

操作允许权	八进制数
用户可读	00400
用户可写	00200
小组可读	00040
小组可写	00020
其他可读	00004
其他可写	00002

控制命令	值
IPC_CREAT	0001000
IPC_EXCL	0002000

如 shmid=shmget(key,size,(IPC_CREAT|400));

表示创建一个关键字为 key，长度为 size 的共享存储区，该存储区为用户可读权限。存储区标识符为 shmid。

（2）shmat(id,addr,flag)。

从逻辑上将一个共享存储区附接到进程的虚拟地址空间上。

该函数调用使用头文件如下：

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
```

参数定义：

```
char *shmat(id,addr,flag)
int id, flag;
char *addr;
```

语法格式：viraddr=shmat(id,addr, flag)

其中，id 是共享存储区的标识符，addr 是用户要使用共享存储区附接的虚地址，若 addr 是 0，则系统选择一个适当的地址来附接该共享区。flag 规定对此区的读写权限以及系统是否应对用户规定的地址做舍入操作。如果 flag 中设置了 shm_ rnd，即表示操作系统在必要时舍去这个地址。如果设置了 shm_rdonly，即表示只允许读操作。viraddr 是附接的虚地址，它的类型为字符指针。

（3）shmdt(addr)。

把一个共享存储区从指定进程的虚地址空间断开。

调用该函数包含以下头文件：

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/mhm.h>
```

参数格式：

```
int shmdt(addr)
char *addr
```

其中，当调用成功时，返回 0 值，调用不成功，返回-1；addr 是系统调用 shmat 所返回的地址。

（4）shmctl(id,cmd,buf)。

对与共享存储区关联的各种参数进行操作，从而对共享存储区进行控制。

调用该函数要包含以下头文件：

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
```

参数格式：

```
int shmctl(id,cmd,buf)
int id,cmd;
struct shmid_ds  *buf;
```

其中，调用成功时返回 0，否则返回-1。id 为被共享存储区的标识符。cmd 规定操作的类型。

- IPC_STAT：返回包含在指定的 shmid 相关数据结构中的状态信息，并且把它放置在用户存储区中的*but 指针所指的数据结构中。执行此命令的进程必须有读取允许权。

- IPC_SET：对于指定的 shmid，为它设置有效用户和小组标识及操作存取权。
- IPC_RMID：删除指定的 shmid 以及与它相关的共享存储区的数据结构。
- SHM_LOCk：在内存中锁定指定的共享存储区，必须是超级用户才可以进行此项操作。

Buf 是一个用户级数据结构地址。它的类型定义为：

```
shmid_ds
{struct ipe--perm shm-perm;        /*许可权结构*/
int shm_segsz;                     /*段大小*/
int padl ;                         /*由系统使用*/
ushort shm_lpid;                   /*最后操作的进程 id*/
ushort shm_cpid ;                  /*创建者的进程 id*/
ushort shm_nattch;                 /*当前附界数*/
short pad2;                        /*由系统使用*/
time_t shm_atime ;                 /*最后附接时间*/
time-t shm-dtime ;                 /*最后断接时间*/
time_t  shm-ctime;                 /*最后修改时间*/
}
```

2.6.2 消息传递系统

不论是单机系统、多机系统还是计算机网络，消息传递机制都是应用最为广泛的一种进程间通信的机制。在消息传递系统中，进程间的数据交换是以格式化的消息（Message）为单位的；在计算机网络中，又把 Message 称为报文。程序员直接利用系统提供的一组通信命令进行通信。操作系统隐藏了实现通信的细节，提高了透明性，因而获得了较为广泛的使用。消息传递系统的通信方式属于高级通信方式，又因实现方式的不同分为直接通信方式和间接通信方式。

1. 直接通信方式

这种通信固定在一对进程之间。例如，进程 A 把信件只发送给进程 B，而进程 B 只接收进程 A 的信件，系统提供两条原语 send 和 receive，用来发送和接收消息。两条原语的形式如下：

```
send(B,message);   /*发送一个消息给接收进程 B*/;
receive(A,message);   /*接收进程 A 发来的消息*/。
```

通常情况下，接收进程可与多个发送进程通信，因此，它不能事先指定发送进程。对于这样的应用，接收进程接收消息的原语中的源进程参数是完成通信后的返回值，接收原语可表示为：

```
receive(id,message);
```

其中，id 为接收消息进程的标识符。

2. 间接通信方式

间接通信方式又称为信箱通信方式。信箱是一种数据结构，逻辑上可分为信箱头和信箱体两部分。信箱头包含信箱体的结构信息，例如，所有格子是构成结构数组还是构成链，以及多进程共享信箱体时的同步互斥信息。信箱体由多个格子构成，它实际上就是一个有

界缓冲池，它的同步、互斥方式与生产者—消费者问题的方式类似。信箱通信一般是进程之间的双向通信，如图 2-15 所示。

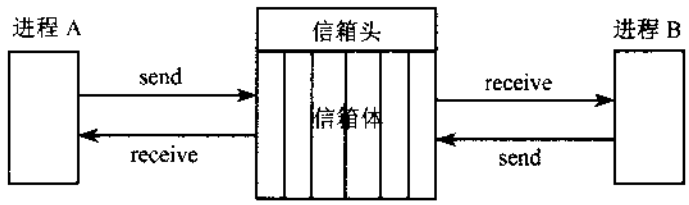


图 2-15　进程的信箱通信方式

3. 消息缓冲队列通信机制

（1）消息缓冲队列通信机制中所用的数据结构。消息缓冲队列通信机制中所用的主要数据结构是消息缓冲区。消息缓冲区作为进程间通讯的一个基本单位。它的描述如下：

```
type message buffer=record
     sender; 发送进程标识符
     size; 消息长度
     text; 消息正文
     next; 指向下一个消息缓冲区的指针
end
```

在设置消息缓冲队列时，还应添加用于对消息队列进行操作和实现同步的信号量，并将它们存入进程的 PCB 中。在 PCB 中增加的数据项描述如下：

```
type processcontrol block=record
     mq; 消息队列队首指针
     mutex; 消息队列互斥信号量
     sm; 消息队列资源信号量
     …
     end
```

当一个发送进程要发送消息时，便形成一个消息，并发送给指定的接收进程。由于接收进程可能会收到几个进程发来的消息，故应将所有消息缓冲区链成一个队列，其队首由接收进程 PCB 中的队列队首指针 mq 指出。

（2）发送原语。发送进程在发送消息之前，应先在自己的内存空间中设置一发送区，把待发送的消息正文、发送进程标识符、消息长度等信息填入其中，然后调用发送原语，把消息发送给接收进程。发送原语首先根据发送区中所设置的消息长度来申请一缓冲区，然后，把发送区中的信息复制到缓冲区 i 中。为了能将缓冲区挂在接收进程的消息队列上，应先获得接收进程的内部标识符 j，然后将 i 挂在 j.mq 上。因为该队列属于临界资源，在执行挂接操作的前后，都要执行 wait 和 signal 操作。

（3）接收原语。接收进程调用接收原语，从自己的消息缓冲队列中选取第一个消息缓冲区，并将其中的数据复制到指定的消息接收区内。

4. Linux 系统关于消息传递的相关系统调用

（1）msgget(key,flag)。

功能：获得一个消息的描述符，该描述符指定一个消息队列以便用于其他系统调用。

该函数使用头文件如下：

```
#include<sys/type.h>
#include<sys/ipc.h>
#include<sys/msg.h>
```

参数定义：

```
int msgget (key,flag);
key_tkey;
int flag;
```

语法格式：

```
msgqid=msgget(key,flag)
```

其中，msgqid 是该系统调用返回的描述符，失败则返回-1；flag 本身由操作允许权和控制命令值相“或”得到。

如 IPC-CREAT|0400，是否该队列应被创建；IPC_EXCL|0400，是否该队列的创建是互斥的等。

（2）msgsnd(id,msgp,size,flag)。

功能：发送一消息。

该函数使用头文件如下：

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
```

参数定义：

```
int msgsnd(id, msgp, size, flag);
int id, size, flag;
struct msgbuf *msgp;
```

其中，id 是返回消息队列的描述符；msgp 是指向用户存储区的一个构造体指针；size 指示由 msgp 指向的数据结构中字符数组的长度，即消息的长度，这个数组的最大值由 MSG-MAX 系统可调用参数来确定。flag 规定当核心用尽内部缓冲空间时应执行的动作；若在标志 flag 中未设置 IPC_NOWAIT 位，则当该消息队列中的字节数超过某一最大值或系统范围的消息数超过某一最大值时，调用 msgsnd 进程睡眠。若是设置 IPC_NOWAIT，则在该情况下，msgsnd 立即返回。

（3）msgrcv(id,msgp,size,type,flag)。

功能：接收一消息。

该函数使用头文件如下：

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
```

参数定义：

```
int msgrcv(id,msgp, size, flag);
int id, size, type, flag;
struct msgbuf  *msgp ;
```

```
struct msgbuf{long  mtype;char mtext[ ];};
```

语法格式：

```
count =msgrcv(id,msgp, size, type, flag)
```

其中，id 是消息描述符；msgp 用来存放要接收消息的数据结构的地址；size 是 msgp 中数据数组的大小；type 是用户要读的消息类型。

- type 为 0：接收该队列的第一个消息；
- type 为正：接收类型 type 的第一个消息；
- type 为负：接收小于或等于 type 绝对值的最低类型的第一个消息。

flag 规定倘若该队列无消息，核心应当做什么事。如果此时设置了 IPC_NOWAIT 标志，则立即返回，若在 flag 中设置了 MSG_NOERROR，且所接收的消息大小大于 size，核心截断所接收的消息。

count 是返回消息正文的字节数。

（4）msgctl(id,cmd,buf)。

功能：查询一个消息描述符的状态，设置它的状态及删除一个消息描述符。

调用该函数使用的头文件如下：

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
```

参数定义：

```
int msgctl(id,cmd,buf)
int id,cmd;
struct msqid-ds  *buf;
```

其中，函数调用成功时返回 0，调用不成功时返回-1。

id 用来识别该消息的描述符；cmd 规定命令的类型。

- IPC_STAT：将与 id 相关联的消息队列首标志读入 buf。
- IPC_SET：为这个消息序列设置有效的用户和小组标识及操作允许权和字节的数量。
- IPC_RMID：删除 id 的消息队列。

buf 是含有控制参数或查询结果的用户数据结构的地址。

msgid_ds 结构定义如下：

```
struct msgid_ds
{struct ipe_perm msg_perm;      /*许可权结构*/
short padl [7] ;                /*由系统使用*/
ushort onsg_qnum;               /*队列上消息数*/
ushort msg_qbytes               /*队列上最大字节数*/
ushort msg_lspid;               /*最后发送消息的 PID*/
ushort msg_lrpid;               /*最后接收消息的 PID*/
time_t msg_stime ;              /*最后发送消息的时间*/
time_t msg_rtime;               /*最后接收消息的时间*/
time_t msg_ctime;               /*最后更改时间*/
};
```

```
struct ipc_perm
{ushort uid;                    /*当前用户 id*/
ushort gid;                     /*当前进程组 id*/
ushort cuid;                    /*创建用户 id*/
ushort cgid;                    /*创建进程组 id*/
ushort mode ;                   /*存取许可权*/
{short padl; long pad2}         /*由系统使用*/
};
```

2.6.3 管道通信系统

管道是指用于连接一个读进程和一个写进程，以实现它们之间通信的一个共享文件，又名 pipe 文件。向管道（共享文件）提供输入的发送进程（写进程），以字符流形式将大量的数据送入管道；而接收管道输出的接收进程（读进程），则从管道中接收（读）数据。由于发送进程和接收进程是利用管道进行通信的，故又称为管道通信。这种方式首创于 UNIX 系统，由于它能有效地传送大量数据，因而又被引入到 Linux 等许多其他操作系统中。为了协调双方的通信，管道机制必须提供以下 3 方面的协调能力。

（1）互斥，即当一个进程正在对 pipe 执行读/写操作时，其他进程必须等待。

（2）同步，指当写（输入）进程把一定数量的数据写入 pipe，便去睡眠等待，直到读（输出）进程取走数据后，再把它唤醒。当读进程读一空 pipe 时，也应睡眠等待，直至写进程将数据写入管道后才将之唤醒。

（3）确定对方是否存在，只有确定对方已存在时，才能进行通信。

2.6.4 信号通信机制

1. 信号的基本概念

每个信号都对应一个正整数常量，称为 signal number，即信号编号。它定义在系统头文件<signal.h>中，代表同一用户的各进程之间传送事先约定的信息的类型，用于通知某进程发生了某异常事件。每个进程运行时，都要通过信号机制来检查是否有信号到达。若有，便中断正在执行的程序，转向与该信号相对应的处理程序，以完成对该事件的处理；处理结束后再返回到原来的断点继续执行。信号机制实际是对中断机制的一种模仿，因此，在早期的 UNIX 版本中又把它称为软中断。

信号与中断的相似点如下：

（1）采用的异步通信方式是相同的。

（2）当有中断请求或检测出有信号时，都暂停正在执行的程序而转去执行相应的处理程序。

（3）都在处理完毕后返回到原来的断点。

（4）对信号或中断都可进行屏蔽。

信号与中断的区别如下：

（1）中断有优先级，而信号没有优先级，所有信号都是平等的。

（2）信号处理程序是在用户态下运行的，而中断处理程序是在核心态下运行。

（3）中断响应是及时的，而信号响应通常都有较大的时间延迟。

信号机制具有以下 3 方面的功能：

（1）发送信号。发送信号的程序用系统调用 kill()实现。

（2）预置对信号的处理方式。接收信号的程序用 signal()实现预置处理方式。

（3）收受信号的进程按事先的规定完成对相应事件的处理。

2. 信号的发送

信号的发送是指由发送进程把信号送到指定进程的信号域的某一位上。如果目标进程正在一个可被中断的优先级上睡眠，核心便将它唤醒，发送进程就此结束。一个进程可能在其信号域中有多个位被置换，代表有多种类型的信号到达，但对于一类信号，进程却只能记住其中的某一个。

进程用 kill()向一个进程或一组进程发送一个信号。

3. 信号的处理

当一个进程要进入或退出一个低优先级睡眠状态或一个进程即将从核心态返回用户态时，操作系统内核检查该进程是否已收到软中断。当进程处于核心态时，即使收到软中断也不予理睬；只有返回到用户态后，才处理软中断信号。对软中断信号的处理分 3 种情况进行：

（1）如果进程收到的软中断是一个已决定要忽略的信号（function=1），进程不做任何处理便立即返回。

（2）进程收到软中断后便退出（function=0）。

（3）执行用户设置的软中断处理程序。

4. 相关的 Linux 系统调用

（1）kill()。

功能：向一个或一组进程发送一个软中断信号。

系统调用格式：

```
int kill(pid,sig)
```

参数定义：

```
int pid,sig;
```

其中，pid 是一个或一组进程的标识符，参数 sig 是要发送的软中断信号。

- pid>0 时，核心将信号发送给进程 pid。
- pid=0 时，核心将信号发送给与发送进程同组的所有进程。
- pid=-1 时，核心将信号发送给所有用户标识符真正等于发送进程的有效用户标识号的进程。

（2）signal()。

功能：预置对信号的处理方式，允许调用进程控制软中断信号。

系统调用格式：

```
signal(sig,function)
```

头文件为：

```
#include <signal.h>
```

参数定义：

```
int  sig;
void (*func) ()
```

其中，sig 用于指定信号的类型，sig 为 0 表示没有收到任何信号，其他取值见表 2-1 所示。

表 2-1 sig 的取值及意义

值	名字	说　明
01	SIGHUP	挂起（hangup）
02	SIGINT	中断，当用户从键盘按^c 键或^break 键时
03	SIGQUIT	退出，当用户从键盘按 quit 键时
04	SIGILL	非法指令
05	SIGTRAP	跟踪陷阱（trace trap），启动进程，跟踪代码的执行
06	SIGIOT	IOT 指令
07	SIGEMT	EMT 指令
08	SIGFPE	浮点运算溢出
09	SIGKILL	杀死、终止进程
10	SIGBUS	总线错误
11	SIGSEGV	段违例（segmentation violation），进程试图访问其虚地址空间以外的位置
12	SIGSYS	系统调用中参数错，如系统调用号非法
13	SIGPIPE	向某个非读管道中写入数据
14	SIGALRM	闹钟。当某进程希望在某时间后接收信号时发此信号
15	SIGTERM	软件终止（software termination）
16	SIGUSR1	用户自定义信号 1
17	SIGUSR2	用户自定义信号 2
18	SIGCLD	某个子进程死
19	SIGPWR	电源故障

function：该进程中的一个函数地址，在核心返回用户态时，它以软中断信号的序号作为参数调用该函数，对除了信号 SIGKILL、SIGTRAP 和 SIGPWR 以外的信号，核心自动地重新设置软中断信号处理程序的值为 SIG_DFL，一个进程不能捕获 SIGKILL 信号。

function=1 时，进程对 sig 类信号不予理睬，即屏蔽了该类信号。

function=0 时，默认值，进程在收到 sig 信号后应终止自己。

function 为非 0、非 1 类整数时，function 的值即作为信号处理程序的指针。

2.7　线程

前面介绍了进程的概念。进程是程序的一次执行，同时也是资源分配的基本单位。自从 20 世纪 60 年代人们提出了进程的概念后，在操作系统中一直都是以进程作为资源分配和独立运行的基本单位。直到 20 世纪 80 年代中期，人们才提出了比进程更小的、能独立运行的基本单位——线程（Threads）。线程比进程能更好地提高程序的并行执行速度，充分地发挥多处理机的优越性，因而当代所推出的多处理机操作系统，如 Windows NT、Windows 2000、OS/390、SunSolaris、Linux、Mach 等操作系统中，都引入了线程的概念，以改善操作系统的性能。

2.7.1　线程的基本概念

引入线程主要是为了提高系统的执行效率，减少处理机的空转时间和在进行调度切换时因保护现场信息所用的时间，便于系统管理。

线程（Threads）是进程中执行运算的最小单位，即执行处理机调度的基本单位。在引入线程的操作系统中，可以在一个进程内部进行线程切换，现场保护工作量小。一方面通过共享进程的基本资源而减轻系统开销，另一方面提高了现场切换的效率，因此，一个进程内的基本调度单位称为线程或轻型线程，这个调度单位既可以由操作系统内核控制，也可以由用户程序控制。下面通过线程与进程的比较，可以进一步理解线程的概念。

（1）进程是资源分配的基本单位。所有与该进程有关的资源，如外部设备、缓冲区队列等，都被记录在进程控制块 PCB 中，以表示该进程拥有这些资源。同一进程的所有线程共享该进程的所有资源。

（2）线程是分配处理机的基本单位，它与资源分配无关，即真正在处理机上运行的是线程。

（3）一个线程只能属于一个进程，而一个进程可以有多个线程，但至少有一个线程。

（4）线程在执行过程中需要协作同步。不同进程的线程间要利用消息通信的方法实现同步。

Linux 系统中基本没有区分进程和线程，它们都使用相同的描述方法，使用相同的调度和管理策略。

2.7.2　线程的状态与转换操作

线程同进程一样，也有自己的状态。线程有 3 种基本状态，即执行、阻塞和就绪，但没有进程中的挂起状态。因此，线程是一个只与内存和寄存器相关的概念，它的内容不会因为交换而进入内存。

针对线程的 3 种基本状态，存在 5 种基本操作来转换线程的状态。

1. 派生（Spawn）

线程在进程中派生出来，也可再派生线程。用户可以通过相关的系统调用派生自己的线

程。在 Linux 系统中，库函数 clonc()和 creat_thread()分别用来派生不同执行模式下的线程。

一个新派生出来的线程具有相应的数据结构指针和变量，这些指针和变量作为寄存器上下文放在本线程的寄存器和堆栈中。新派生出来的线程被放入就绪队列。

2. 调度（Schedule）

选择一个就绪线程使之进入执行状态。

3. 阻塞（Block）

像进程一样，如果一个线程在执行过程中需要等待某个事件发生，则被阻塞。阻塞时，寄存器上下文、程序计数器以及堆栈指针都会得到保证。

4. 激活（Unblock）

如果阻塞线程所等待的事件发生，则该线程被激活并进入就绪队列。

5. 结束（Finish）

如果一个线程执行结束，它的寄存器上下文以及堆栈内容等将被释放。

Linux 的系统级线程在表示格式、管理调度等方面与进程没有严格的区分，都是当作进程来统一对待。

2.7.3 引入线程的好处

（1）易于调度。由于线程只作为独立调度的基本单位，同一进程的多个线程共享进程的资源，所以线程易于切换。

（2）提高了系统的效率。通过线程可以方便有效地实现并发性。进程可创建多个线程来执行同一程序的不同部分。

（3）创建一个线程比创建一个进程花费的开销少，创建速度快。

（4）有利于发挥多处理器的功能，提高进程的并行性。

2.7.4 多线程的实现

多线程机制是指操作系统支持在一个进程内执行多个线程的能力。从线程的观点分析，MS-DOS 仅支持一个用户进程和一个线程；UNIX 系统支持多个用户进程，但一个进程只有一个线程；Windows NT、Solaris、Linux 等支持多进程多线程。

虽然多种系统都支持多线程，但实现的方式并不完全相同。线程有两个基本类型：用户级线程和系统级线程（也称核心级线程）。有的系统实现的是用户级线程，如 Informix 数据库系统；有些系统实现的是内核级线程，如 Mach 操作系统；有些系统同时实现这两种类型的线程，如 Linux 和 Sun 公司的 Solaris 操作系统。

1. 用户级线程

用户级线程简称为 ULT，是由用户应用程序建立的，并由用户应用程序负责对这些线程进行调度和管理，操作系统内核并不知道有用户级线程的存在，只对进程进行管理。因而这种线程与内核无关。这是通常所说的“纯 ULT 方法”，MS-DOS 和 UNIX 操作系统就属于此类。

这种纯 ULT 方法的优点如下：

（1）应用程序中线程开关的时空开销远小于内核级线程的开销。

（2）线程的调度算法与操作系统的调度算法无关。

（3）用户级线程方法适用于任何操作系统，因为它与内核无关。

缺点为：

（1）在一个典型的操作系统中，有许多系统请求正被阻塞着，因此，当线程执行一个系统请求时，不仅本线程阻塞，而且该进程中所有线程都被阻塞。

（2）在该方法的系统中，因为每个进程每次只能由一个线程在 CPU 上运行，因此，一个多线程应用无法利用多处理器的优点。

2. 内核级线程

内核级线程简称为 KLT，通常也称为“纯 KLT”方法。内核级线程中所有线程的创建、调度和管理全部由操作系统内核负责完成。一个应用进程可按多线程方式编写程序，当它被提交给多线程操作系统运行时，内核为它创建一个进程和一个线程，线程在运行中还会创建新的线程。操作系统内核给应用程序提供相应的系统调用和应用程序接口，以使用户程序可以创建、执行、撤消线程。Windows NT 属于此类。

这种内核级线程的优点为：

（1）内核可调度一个进程中的多个线程，使其同时在多个处理机上并行运行，从而提高系统的效率。

（2）当进程中的一个线程被阻塞时，进程中的其他线程仍可运行。

（3）内核本身可以以线程方式实现。

缺点为：由于线程调度程序运行在内核态，而应用程序运行在用户态，因此同一进程中的线程切换要经过从用户态到核心态，再从核心态到用户态的两次模式转换。

3. 用户级线程与核心态线程相结合的模式

由于用户级线程和内核级线程各有特色，因此，如果将两种方法结合起来，则可吸取两者的优点，结合起来的系统称为多线程的操作系统。内核支持多线程的建立、调度和管理。同时系统中又提供使用线程库，允许用户应用程序建立、调度和管理用户级线程。

Linux 的内核级线程和其他操作系统的内核实现不同。大多数操作系统单独定义描述线程的数据结构，采用独立的线程管理方式，提供专门的线程调度，这些都增加了内核和调度程序的复杂性。而在 Linux 中，将线程定义为“执行上下文”，它实际只是进程的另外一个执行上下文而已，和进程采用同样的表示、管理、调度方式。这样，Linux 内核并不需要区分进程和线程，只需要一个进程/线程数组，而且调度程序也只有进程的调度程序，内核的实现相对简单得多，而且节约系统用于管理方面的时间开销。

本章小结

进程是操作系统中一个非常重要的概念。进程是程序的一次执行，同时也是操作系统进行资源分配的单位。进程具有一些特征，是与程序有根本区别的概念。进程具有动态性、并发性、异步性、独立性的特性。反映进程动态性的是进程状态的变化。进程从创建到被

撤消，要经过一些具有生命状态的活动。进程的3个基本状态包括阻塞、就绪、执行，除此之外，不同的操作系统还具有其他一些状态。进程的状态转换由相应的原语来完成。进程的并发执行是指在同一时间间隔内多个进程同时发生。进程的并发特性反映在进程对资源的竞争以及由资源竞争所引起的对进程执行速度的制约。可以通过提高进程的并发性来提高整个系统的效率。

一个进程的静态描述是处理机的一个执行环境，称为进程上下文。在Linux系统中，子进程继承父进程的进程上下文。进程上下文由以下部分组成：进程控制块（PCB）、正文段（或程序）、数据段以及各种寄存器和堆栈中的值。进程控制块PCB是进程存在的惟一标志，它包含进程的运行信息和程序的控制信息。进程控制块在内存中的组织方式有线性方式、链接方式和索引方式。对于Linux系统，可以通过几个常用的进程创建和控制的系统调用实现对进程的控制。

不能被多个进程同时使用的资源称为临界资源。将每个进程中访问临界资源的那段代码称为临界区。多个进程不能同时进入同一个临界区，这称为进程之间的互斥；多个进程在使用临界资源时，表现出来的相互协调、相互合作、互相等待，使得各进程按一定速度执行的过程称为进程间的同步。具有同步关系的一组并发进程称为合作进程。实现进程的互斥和同步，可以用锁或P、V操作来实现。P操作是申请临界资源的原语操作，也称为wait操作，V操作是释放临界资源的原语操作，也称为signal操作。信号量是P、V操作的对象。

另一个重要的概念是进程通信。进程通信可包括低级通信和高级通信。对进程进行控制的通信是低级通信；传递大量数据的通信称为高级通信。高级通信包括共享存储器系统、消息传递系统和管道通信系统。

与进程概念密切相关的概念是线程。线程可看成是进程中指令的不同执行路线，它是为了提高操作系统的执行效率而引入的。线程又称为轻型进程，在有线程的操作系统中，它是操作系统分配处理机的基本单位，而进程是分配资源的基本单位。线程分为用户线程和内核线程。

习 题 二

2-1 操作系统为什么要引入进程的概念？

2-2 试比较进程和程序的区别。

2-3 程序并发执行，为什么会失去封闭性和再现性？

2-4 什么叫进程的并发性？试举一个进程并发执行的例子。

2-5 举一个例子，说明一个程序可能同时属于多个进程。

2-6 试说明PCB的作用，为什么说PCB是进程存在的惟一标志？

2-7 说明进程由哪几部分构成？

2-8 什么叫临界区？为什么进程在进入临界区之前要先执行申请操作，离开临界区要做释放操作？

2-9　试说明进程的基本状态及转换的原因。

2-10　在创建一个进程时，所要做的工作有哪些？

2-11　从概念上说明记录型信号量的构成，描述原语 wait 和 signal 所进行的操作。

2-12　在生产者—消费者问题中，如果缺少了 signal(full)或 signal(empty)，对执行结果将会有何影响？

2-13　在生产者—消费者问题中，如果两个 wait 操作即 wait(full)和 wait(empty)位置互换，会产生什么后果？

2-14　进程的高级通信方式有哪几种？

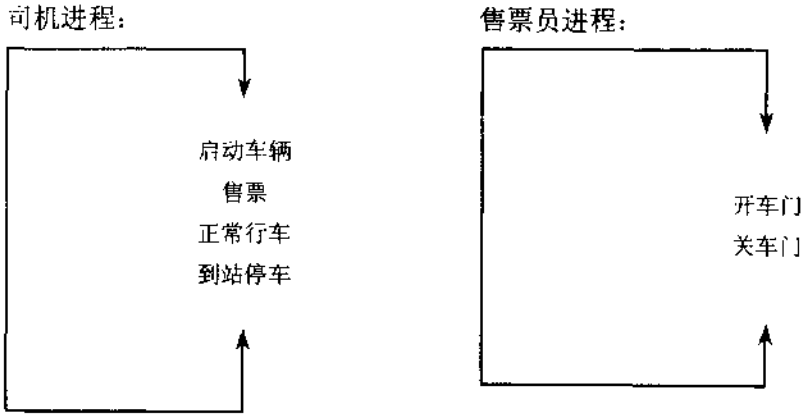
2-15　什么是线程？说明它与进程的主要区别。

2-16　什么是多线程机制？引入它有什么好处？

2-17　在读者—写者问题中，如果修改问题中的同步算法，要求对写进程优先，即一旦写进程到达，后续的读者进程必须等待，而不管是否有读者进程在读文件。写出相应进程的程序段。

2-18　试利用记录型信号量写出一个不会出现死锁的哲学家进餐问题的算法。

2-19　设公共汽车上有一位司机和一位售票员，它们的活动如下：



请分析司机与售票员之间的同步关系，并用 P、V 操作实现。

第 3 章　处理机调度与死锁

本章学习目标

本章主要介绍计算机系统中处理机的三级调度，作业调度、进程调度的概念及算法，同时加以应用。并发执行的各进程在使用资源时，可能出现死锁问题，本章分析了死锁的原因及处理方法。通过本章的学习，读者应掌握以下内容：

- 处理机的三级调度
- 作业调度及进程调度的概念
- 调度算法的评价准则
- 掌握并灵活运用常用的几种作业调度、进程调度算法
- 死锁的概念、产生的原因及死锁的必要条件
- 死锁的预防方法及利用银行家算法避免死锁的方法
- 死锁的检测与恢复的方法，并能灵活运用

3.1　分级调度

操作系统一个非常重要的功能就是管理计算机资源，提高系统的效率。对处理机的管理是操作系统的基本功能之一。在早期的计算机系统中，对 CPU 的管理是十分简单的，因为那时它和其他系统资源一样，被一个作业所独占，不存在处理机分配和调度问题。随着多道程序设计技术和各种类型的操作系统的出现，各种不同的 CPU 管理方式开始使用，为用户提供了不同性能的操作系统。例如，在多道批处理系统中，为了提高处理机的效率和增加作业吞吐率，当调度一批作业，组织多道运行时，要尽可能使作业组织合理，但由于在批处理系统中，一旦把作业提交给计算机系统，直到作业运行完成，用户都不能插手干预自己的作业，而且，作业的响应时间一般都较长，因此，在用户看来，这是一台没有交互、速度较慢的处理机。但是，在批处理系统中，其资源的利用率和系统的吞吐量高。在分时系统中，由于用户使用交互式会话的工作方式，系统必须有较快的响应时间，使得每个用户都感到如同只有他自己一人在使用这台机器，因此，系统在调度作业执行时，要首先考虑每个用户作业得到处理机的均等性。这样，系统资源的利用率就不如批处理系统。由此可以看出，根据操作系统的要求不同，处理机管理的策略也是不同的。

一个批处理型作业，从进入系统并驻留在外存的后备队列上开始，直至作业运行完毕，可能要经历以下三级调度，即作业调度、对换和进程调度。

3.1.1　调度的层次

1. 作业调度

作业调度又称高级调度或长调度，用于选择把外存上处于后备队列中的哪些作业调入内存，并为它们创建进程、分配必要的资源。然后，再将新创建的进程排在就绪队列上，准备执行。在批处理系统中，作业进入系统后，是先驻留在外存上的，因此需要有作业调度的过程，以便将它们分批地装入内存。在分时系统中，为了做到及时响应，用户通过键盘输入的命令或数据等，都是被直接送入内存的，因而无须再配置作业调度机制。类似地，在实时系统中，通常也不需要作业调度。

一个作业从提交给计算机系统到执行结束，退出系统，一般要经历提交、后备、执行和完成 4 个状态。

（1）提交状态。一个作业在处于从输入设备进入外部存储设备的过程称为提交状态。处于提交状态的作业，因其信息尚未全部进入系统，所以不能被调度程序选中。

（2）后备状态。也称为收容状态。输入管理系统不断地将作业输入到外存中对应部分（或称输入井）。若一个作业的全部信息已全部被输入到输入井，则在它还未被调度执行之前，该作业处于后备状态。

（3）执行状态。作业调度程序从后备作业中选取若干个作业到内存投入运行，及为被选中作业建立进程并分配必要的资源，这时，这些被选中的作业处于执行状态。从宏观上看，这些作业正处在执行过程中，从微观上看，在某一时刻，由于处理机总数少于并发执行的进程数，因此，不是所有被选中作业都占有处理机，其中的大部分处于等待资源或就绪状态。哪个作业的哪个进程被分配给处理机，这是进程调度要完成的任务。

（4）完成状态。当作业运行完毕，但它所占用的资源尚未全部被系统回收时，该作业处于完成状态。在这种状态下，系统需做如打印结果、回收资源等类似的善后处理工作。

2. 对换

对换又称交换调度或中级调度，其主要任务是按照给定的原则和策略，将处于外存交换区中的就绪状态或等待状态的进程调入内存，或把处于内存就绪状态或内存等待状态的进程交换到外存交换区。交换调度主要涉及内存管理与扩充，我们将在内存管理部分讨论这个问题。

3. 进程调度

进程调度又称为低级调度或微观调度，其主要任务是按照某种策略和算法，将处理机分配给一个处于就绪状态的进程。在确定了占用处理机的进程后，系统必须进行进程上下文切换以建立与占用处理机进程相适应的执行环境。进程调度可分为如下两种方式。

（1）非抢占方式。非抢占方式不允许进程抢占已经分配出去的处理机。采用非抢占调度方式时，可能引起进程调度的原因有正在执行的进程执行完成，或因发生某事件而不能继续执行；执行中的进程因提出 I/O 请求而暂停执行；在进程通信或同步过程中执行了某种原语操作，如 P 操作（wait 操作）、Block 原语、Wakeup 原语等。

非抢占调度方式的优点是实现简单、系统开销小，适用于大多数的批处理系统环境。

但它很难满足紧急任务的要求，因而可能造成难以预料的后果。显然，在要求比较严格的实时系统中，不宜采用这种调度方式。

（2）抢占方式。抢占调度方式允许调度程序根据某种原则暂停某个正在执行的进程，将处理机收回，重新分配给另一个进程。抢占的原则有优先权原则、短作业（或短进程）优先原则、时间片原则等。

作业调度与进程调度关系如图 3-1 所示。

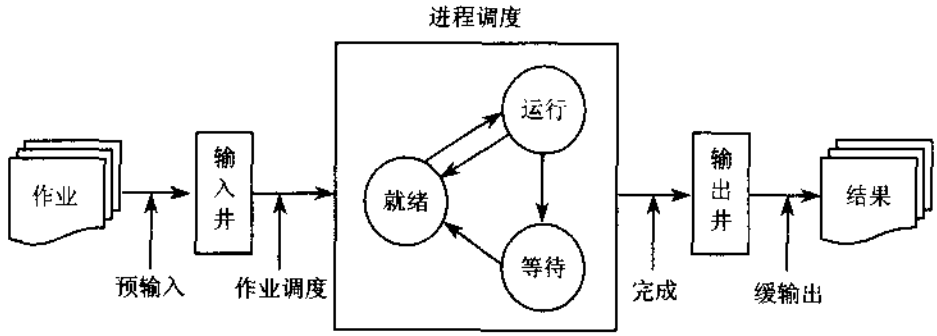


图 3-1 作业调度与进程调度

3.1.2 作业与进程的关系

作业是用户向计算机提交任务的任务实体。一个作业是指在一次应用业务处理过程中，从输入开始到输出结束，用户要求计算机所做的有关该次业务处理的全部工作，如一次计算、一个控制过程等。进程是计算机为了完成用户任务实体而设置的执行实体，是系统分配资源的基本单位。显然，计算机要完成一个任务，必须有一个以上的执行实体，即一个作业总是由一个或多个进程组成的。作业分解为进程的过程是系统首先为一个作业创建一个根进程。然后，在执行作业控制语句时，根据任务要求，系统或根进程为其创建相应的子进程。最后，进行进程调度，为各子进程分配资源和调度各子进程执行，以完成作业要求的工作。

3.2 作业调度

作业调度主要是完成作业从后备状态到执行状态的转换，以及从执行状态到完成状态的转换。本节主要介绍作业调度的功能及调度性能的评价方法。

3.2.1 作业调度的功能

作业调度主要完成的具体功能如下。

1. 记录系统中各作业的状态

作业调度程序要从后备态队列中选择一个作业使它执行，并在执行过程中对其进行管理，这就要求系统必须掌握作业在各个状态，包括执行阶段的有关情况。一般，系统为每个作业建立一个作业控制表 JCB，用以记录这些有关信息。正如系统通过进程控制块 PCB

而感知进程的存在一样，系统通过作业控制块 JCB 而感知作业的存在。系统在作业进入后备状态时，为它建立了作业控制块 JCB，从而使该作业可被作业调度程序选中。当该作业执行完毕进入完成状态后，系统又撤销其 JCB，释放有关资源并撤销该作业。每个作业的状态、在各个阶段所需要和已分配的资源都记录在它的 JCB 中，根据 JCB 中的有关信息，作业调度程序对作业进行调度和管理。

对于不同的批处理系统，其 JCB 的内容也有所不同。JCB 的主要内容如图 3-2 所示。

作业名	
作业类型	计算型 管理型 图形设计型
资源要求	内存量 外存量 外设类型及数量 软件支持工具库函数
当前状态	提交状态 后备态 运行态 完成
资源使用情况	进入系统的时间 开始执行时间 已运行时间 内存地址 外设台数
作业的优先级	

图 3-2　作业控制块 JCB

从图 3-2 可以看出，JCB 的主要功能如下。

作业名：由用户提供并由系统将其转换为系统可识别的作业标识符。

作业类型：指该作业属于计算型（要求 CPU 时间多）、管理型（要求输入/输出量大）或图形设计型（要求高速图形显示）等。

资源要求：要求的内存量和外存量、外设类型及数量，以及要求的软件支持工具库函数等。资源要求均由用户提供。

当前状态：是指该作业当前所处的状态。显然，只有当作业处于后备状态时，该作业才可以被调度。

资源使用情况：包括作业进入系统的时间、开始执行时间、已执行时间、内存地址、外设数量等。作业进入系统时间是指作业的全部信息进入输入并，作业的状态为后备状态的时间。开始执行时间指该作业被调度程序选中，其状态由后备状态变为执行状态的时间。

内存地址指分配给该作业的内存区起始地址。外设台数指分配给该作业的外设实际数量。

作业的优先级：优先级用来决定该作业的调度次序。优先级既可以由用户给定，也可以由系统动态计算产生。

2. 从后备队列中挑选出一部分作业投入执行

一般来说，系统中处于后备状态的作业较多，大的系统可以达到几十个甚至几百个，这取决于输入井空间的大小。但是，处于执行状态的作业一般只有有限的几个。作业调度程序根据选定的调度算法，从后备作业队列中挑选出若干作业去投入执行。

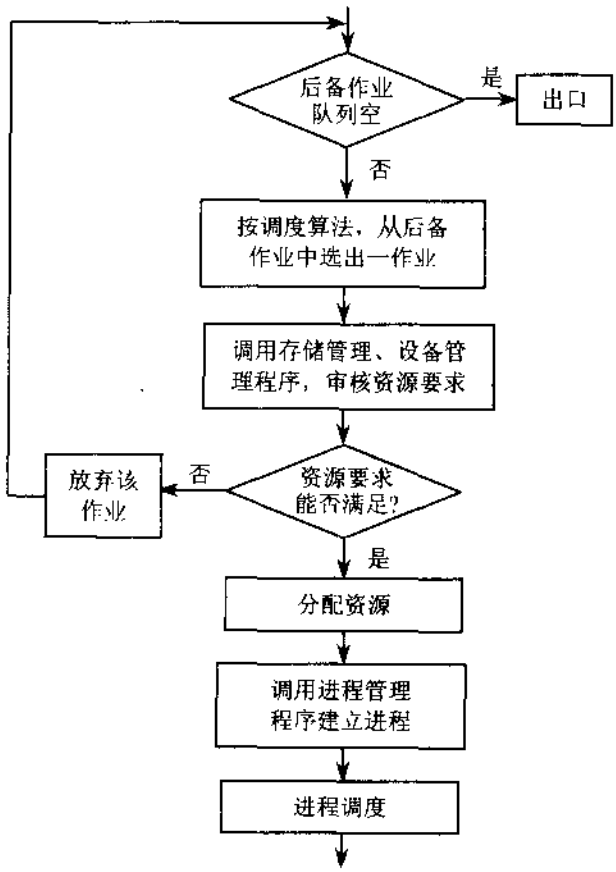
3. 为被选中作业做好执行前的准备工作

作业调度程序为选中的作业建立相应的进程，并为这些进程分配它们所需要的系统资源，如分配给它们内存、外存、外设等。

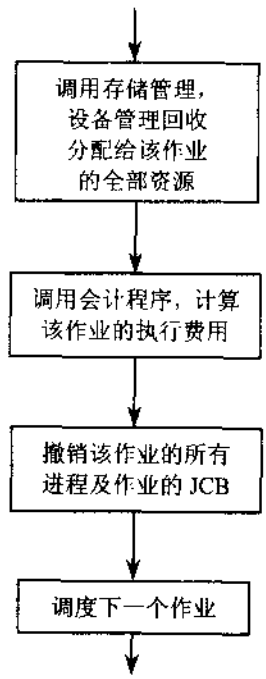
4. 在作业执行结束时做好善后处理工作

包括输出作业管理信息，如执行时间等；回收该作业所占用的资源；撤销与该作业有关的全部进程和该作业的作业控制块等。

作业从后备状态到执行状态以及从执行状态到完成状态的转换过程如图 3-3 所示。



（a）作业从后备状态到执行状态



（b）作业从执行状态到完成状态

图 3-3 作业的状态转换过程

Linux 系统的作业一旦输入，就直接进入内存，建立相应的进程，进入下一级的调度。因此，Linux 系统没有作业调度的概念。

3.2.2　调度算法的评价及准则

在操作系统的设计中，如何选择作业调度及进程调度的方式和算法取决于操作系统的类型及目标。作业调度的功能是记录系统中各作业的状况；从后备作业队列中挑选一批作业进入执行状态；为被选中作业分配资源建立进程，并在作业执行结束后释放所占用的资源等。其中最主要的功能是从后备作业队列中选取一批作业进入执行状态。显然，根据不同的目标，会有不同的调度算法。我们在这里介绍调度的目标。

1. 面向用户的准则

这是面向用户所需遵循的一些准则。其中，比较重要的有以下几点。

（1）周转时间短。通常把周转时间的长短作为衡量批处理系统的性能、选择作业调度方式与算法的重要准则之一。

周转时间：作业被提交给系统开始，到作业终止为止的这段时间间隔，也称为作业周转时间，它包括作业在外存后备队列上等待调度的时间；进程在就绪队列上等待进程调度的时间；进程占用 CPU 执行的时间；进程等待 I/O 操作完成的时间。对于一个用户来说，作业周转时间越短越好。

作业 i 的周转时间 T_i 可定义为：

$$T_i=T_{ei}-T_{si}$$

其中，T_{ei} 为作业 i 的完成时间，T_{si} 为作业 i 的提交时间。

作为计算机系统的管理者，总是希望平均周转时间最短，这不仅能有效地提高系统资源的利用率，而且还可使大多数用户感到满意。若一作业流含有 n 个作业，则它的平均周转时间为：

$$T=\frac{1}{n}\left[\sum_{i=1}^{n}T_i\right]$$

作业的周转时间 T 与系统为它提供服务的时间（即作业要求运行时间）T_s 之比称为带权周转时间，即带权周转时间为：

$$W=T/T_s$$

因为周转时间 T=等待时间+运行时间，因此，W 也可表示为：

$$W=1+\text{等待时间}/\text{运行时间}$$

从公式可以看出，W 值的变化范围是 $W\geqslant 1$，即 W=1 为 W 取值的最小值。可以看出，带权周转时间越接近 1，该作业相对等待时间越短，系统性能越高。

而平均带权周转时间可表示为：

$$\overline{W}=\frac{1}{n}\left[\sum_{i=1}^{n}\frac{T_i}{T_{si}}\right]$$

（2）响应时间快。常用响应时间的长短来评价分时系统的性能。响应时间是指从用户提交一个作业请求开始，直至系统首次产生响应为止的时间，它包括从键盘输入的请求

信息传送到处理机的时间；处理机执行响应处理的时间；将所形成的响应信息在终端显示器上显示出来的时间。

（3）截止时间的保证。这是选择实时调度算法的重要准则。截止时间是指某任务必须开始执行的最晚时间，或必须完成的最晚时间。

（4）优先权准则。在批处理、分时和实时系统中选择调度算法时都可遵循优权准则，以便使某些紧急作业能够得到及时处理。

2. 面向系统的准则

设计调度算法时应考虑整个系统的效率，具体包括如下因素：

（1）系统的吞吐量。吞吐量是指在单位时间内系统所完成的作业数，它与批处理作业的平均长度有密切关系。

（2）处理机的利用率。对于单用户的微机或某些实时系统，此准则不太重要。

（3）各类资源的平衡利用。尽量使 I/O 繁忙的作业（即输入/输出型作业）与 CPU 繁忙的作业（计算型作业）搭配运行，使 CPU 及各种外设并行执行。

3.3 进程调度

一般情况下，在单 CPU 系统中，处在就绪状态的用户进程数可能不止一个。同时，系统进程也可能需要使用处理机，这就要求进程调度程序按一定策略，动态地把处理机分配给处于就绪队列中的某一个进程，使之执行。本节介绍进程调度的功能、进程调度发生的时机以及进程调度引起的进程上下文切换等。

3.3.1 进程调度的功能

进程调度的功能可总结为如下几个方面。

1. 记录系统中所有进程的执行情况

在进行进程调度以前，进程管理模块已将系统中各进程的执行情况和状态特征记录在各进程的 PCB 中。而且，进程管理模块根据各进程的状态特征和资源需求，将各进程的 PCB 表排成相应的队列并进行动态队列转接。进程调度模块通过各进程 PCB 的变化，掌握系统中所有进程的执行情况和状态特征，并在适当的时候从就绪队列中选出一个进程占据处理机。

2. 从就绪状态队列中选择一个进程

进程调度的主要功能是在就绪状态的进程 PCB 队列中，按照一定的策略选择一个进程，分配给处理机并使之执行。不同的系统设计目的对应有不同的选择策略。例如，静态优先数算法系统开销较少，因此适合于分时系统的轮转法和多级反馈队列轮转法。策略的选择决定了调度算法的性能。

3. 进行进程上下文的切换

一个进程的上下文包括进程的状态、有关变量和数据结构的值、机器寄存器的值和 PCB 以及有关程序、数据等。一个进程的执行必然是在进程的上下文中执行。当正在执行

的进程由于某种原因让出处理机时，系统要做进程上下文切换，以使调度程序新选中的进程得以执行。当进程上下文切换时，系统应首先检查是否允许做上下文切换（有些情况下，上下文切换是不允许的，如系统正在执行某个不允许中断的原语时）；然后，系统要保留有关被切换进程的相关信息，以便切换回该进程时，顺利恢复该进程的执行。在系统保留了 CPU 现场之后，调度程序从就绪状态队列中选择一个进程，并装配该进程的上下文，将 CPU 的控制权交给它。

3.3.2　进程调度的时机

在并发执行的环境下，有如下引起进程调度的事件。

1. 完成任务

正在执行的进程运行完成，主动释放对 CPU 的控制。

2. 等待资源

由于等待某些资源或事件发生，正在运行的进程不得不放弃 CPU，进入阻塞状态。

3. 运行时间到

在分时系统中，当前进程使用完规定的时间片，时钟中断，使该进程让出 CPU。

4. 进入睡眠状态

执行中的进程自己调用阻塞原语将自己阻塞起来，进入睡眠等待状态。

5. 发现标志

执行完系统调用，即核心处理完陷入事件后，从系统程序返回用户进程时，可认为系统进程执行完毕，从而可调度新的用户进程执行。

以上都是在 CPU 执行不可剥夺方式下引起进程调度的原因。当 CPU 执行方式是可剥夺时，引起进程调度的事件除上述 5 种外，还应再加一条：优先级变化，如以下标题 6 所述。

6. 优先级变化

就绪队列中某进程的优先级高于当前执行进程的优先级时，将引起进程调度。

只要上述几种原因之一发生，操作系统就将进行进程调度。

3.3.3　进程上下文的切换

进程的上下文由正文段、数据段、硬件寄存器的内容以及有关的数据结构组成。硬件寄存器主要包括存放 CPU 将要执行的下条指令虚地址的程序计数器、指出机器与进程相关联的硬件状态的处理机状态寄存器 PS，存放过程调用时所传递参数的通用寄存器 R 和堆栈指针寄存器 S 等。数据结构包括 PCB 等在内的所有与执行该进程有关的管理和控制用表格、数组、链等。当进程调度发生时，系统要做进程上下文的切换。进程上下文的切换主要包括以下 4 个方面。

1. 决定是否要做以及是否允许做上下文切换

检查分析进程调度原因，以及当前执行进程的资格和 CPU 执行方式的检查等，在操作系统中，进程上下文切换程序在如上所述时机进行上下文的切换。

2. 保存当前执行进程的上下文

这里所说的当前执行的进程是指要停止执行的进程。如果上下文切换程序不是被那个当前执行进程所调用，且不属于该进程，则所保存的上下文应该是先前执行进程的上下文，以便该进程下次执行时恢复它的上下文。

3. 选择一个处于就绪态的进程

按照某种进程调度算法选择一下处于就绪态的进程。

4. 使被选中的进程执行

恢复被选中进程的上下文，给它分配处理机，使之执行。

3.3.4 Linux 系统中进程调度发生的时机

对于 Linux 系统，没有设置专门的调度进程，需要进程调度时，调用一个特定的调度函数来完成该功能。通常，Linux 系统中进程调度发生的时机有以下几种。

（1）用户创建一个新的进程时，系统把它加到就绪队列中，返回该进程的进程标识号。这时，调度函数开始执行，这样可以保证系统具有很好的响应特性。

（2）正在执行的进程申请资源或等待某个事件的发生，而得不到满足时，该进程进入等待状态；当正在执行的进程完成了任务或得到特定的信号而退出，将转入僵死状态。这两种进程状态转换完成后，调用调度函数，选择新的进程分配给 CPU。

（3）分时系统中，进程执行完一定的时间片后，释放 CPU 资源。

（4）Linux 系统提供了两级保护，用户进程可以在用户态和核心态下运行。具有不同的特权级别，可以访问不同的地址空间。用户进程可以在这两种模式之间进行切换，用户态通过中断或函数调用就可以转入核心态。其中，中断是应进程外部发来信息的要求而转入核心态，函数调用则是应进程内部要求转入核心态，而从核心态切换到用户态，则需要一定的硬件支持。当进程从核心态返回到用户态时，将调用调度函数，发生调度。

从根本上说，这些情况可以归结为两类，一是进程本身自动放弃处理机，发生调度，包括进程转换到等待和僵死状态，这类调度是用户进程可以预测的；二是由核心态转入用户态时发生调度，这类调度发生最为频繁。系统调用完成和内核处理完中断之后，系统由核心态转入用户态，时间片用完是系统发送的时钟中断，本质上也是一种中断，而就绪队列加入新进程的工作只能由内核操作完成，无疑是发生于内核态。

3.4 调度算法

本节讨论各种常见的作业调度算法和进程调度算法。在操作系统中，调度的实质是一种资源分配，因而调度算法是根据系统的资源分配策略，规定资源分配的算法。现有的各种调度算法中，有的适用于作业调度，有的则适用于进程调度，也有一些既适合于作业调度，也适合于进程调度。对于不同的系统和调度目标，应采用不同的调度算法。例如，在批处理系统中，考虑作业的平均周转时间，应采用短作业优先的调度算法；在分时系统中，为了保证系统具有合理的响应时间，应采用轮转法进行调度。

3.4.1　单道批处理系统的调度算法

单道批处理系统中，作业调度的主要任务是解决作业之间的自动接续问题，减少操作员的干预，提高系统资源的利用率。所采用的调度算法比较简单，有如下几种。

1. 先来先服务（FCFS）调度算法

先来先服务（FCFS）调度算法是一种简单的调度算法，它既适用于作业调度，也适用于进程调度。该算法是按照作业或进程到达的先后次序来进行调度。当在作业调度中采用该算法时，每次调度都是从后备队列中选择一个最先进入该队列的作业，将它调入内存，为其创建进程、分配相应的资源，将该作业的进程放入就绪队列。在进程调度中采用该算法时，每次调度是从就绪队列中选择一个最先进入该队列的进程，并给它分配处理机。

FCFS 算法比较利于长作业或进程，而不利于短作业或进程。

例 1：现有 4 个作业，假设它们按顺序依次到达，但到达的前后时间忽略不计。要求运行时间分别为 2 秒、60 秒、2 秒、2 秒，如表 3-1（a）所示。假设作业提交时刻为 0。若系统按照 FCFS 算法进行作业调度，要求计算各作业的开始运行时间、运行结束时间、周转时间和带权周转时间。

通过计算，运算结果如表 3-1（b）所示。

表 3-1　FCFS 算法示例

（a）

作业名	到达次序	运行时间（秒）
A	1	2
B	2	60
C	3	2
D	4	2

（b）

作业名	到达次序	运行时间（秒）	开始执行时间	完成时间	周转时间	带权周转时间
A	1	2	0	2	2	1
B	2	60	2	62	62	1.03
C	3	2	62	64	64	32
D	4	2	64	66	66	33
平均值					48.5	16.76

从表 3-1（b）可以看出，作业 C 的带权周转时间长达 32，作业 D 的带权周转时间也较大，但如果作业 D 的要求运行时间长一些，则它的带权周转时间就会相应地降低。

2. 短作业（进程）优先调度算法

短作业调度算法（SJF）或短进程调度算法（SPF）是指对短作业或短进程优先调度的

算法。这里，作业或进程的长短是以作业或进程要求运行时间的长短来衡量的。

例 2：仍然用例 1 中的 4 个作业，采用 SJF 算法对它们的开始运行时间、运行结束时间、周转时间和带权周转时间进行讨论。对于相同长度的作业，通常按照 FCFS 算法执行。通过计算，可得如表 3-2 所示的结果。

表 3-2 列出了各进程的开始执行时间、完成时间、周转时间和带权周转时间。比较表 3-1（b）和表 3-2 可以看出，短作业优先调度算法对短作业带来明显的改善，同时降低了作业的平均周转时间，提高了整个系统的性能。

表 3-2 SJF 算法示例

作业名	到达次序	运行时间（秒）	开始执行时间	完成时间	周转时间	带权周转时间
A	1	2	0	2	2	1
B	2	60	6	66	66	1.1
C	3	2	2	4	4	2
D	4	2	4	6	6	3
平均值					19.5	1.8

SJ(P)F 调度算法也存在一些不容忽视的缺点。

（1）该算法对长作业不利。若系统不断有短作业进入，将造成长作业无限期延迟得不到调度。

（2）该算法未考虑作业的紧迫度，因而不能保证紧迫作业的及时处理。

（3）由于作业或进程的长短只是由用户估计的，而用户又可能有意无意地缩短作业的估计运行时间，因此不一定保证做到真正意义上的短作业优先调度。

3. 高响应比优先调度算法

高响应比优先调度算法（Highest Response_Ratio Next）又称为 HRN 调度算法。在批处理系统中，短作业优先算法的主要不足是长作业的运行得不到保证。为了克服短作业优先算法的缺点，可以采用一种折衷的方法，既让短作业优先，又考虑到系统内等待时间过长的作业，这就是 HRN 调度算法。该调度策略考虑每个作业等待时间的长短的同时估计所需的执行时间长短，从中选出响应比最高的作业投入运行。响应比 R 可定义为：

R=（已等待时间＋要求服务时间）/要求服务时间

即 R=1＋已等待时间/要求服务时间。

所谓响应比 R 高者优先调度是指每次挑选一个作业投入执行时，先计算此时后备作业队列中每个作业的响应比 R 值，选择一个 R 值最高者投入执行。从上面的公式可以看出，一个作业的响应比随着等待时间的增加而增大，因此，一个长期未被调度的作业，只要等待足够长的时间，总有机会成为响应比高者，从而得以执行；同样长度的作业，其 R 值可以从其等待调度的时间长短来区分；同样等待时间的作业，要求执行时间少的作业 R 值高。因此，该算法既照顾了短作业，又考虑了作业的等待时间。当然，利用该算法时，每次进行调度之前，都要先计算响应比，因此增加了系统的开销。

3.4.2　多道批处理系统的调度算法

在多道程序设计系统中，由于某道作业的输入输出操作由通道代行控制，将 CPU 从繁多的 I/O 操作中解脱出来，因此 CPU 可再分配给另一道作业，从而形成了多道作业并发执行的结果，减少了 CPU 对 I/O 的等待时间。但是，由于作业不同，各作业执行时包含的 I/O 操作时间的比例也不同。安排多少道作业，才能使 CPU 对 I/O 的等待时间最小，是难以确定的。而且，当多道程序同时运行时，难以准确地确定一个作业的完成时间。因此，计算周转时间和带权的周转时间比较困难。但是，总体来看，采用多道程序并发执行肯定会减少作业的周转时间和带权的周转时间，从而使系统的效率得以提高。

由于上述原因，大多数多道程序系统的作业调度采用以下策略：先来先服务；考虑优先级；分时和优先级相结合；综合考虑资源要求等。

1. 基于先来先服务调度

这是一种最简单的调度算法。作业调度程序只需按作业进入的时间顺序建立一个后备作业队列，当需要调度作业时，由作业调度程序从队首开始扫描这个队列，直到选择一个资源能够得到满足的作业，并选中该作业。

2. 基于优先级调度

每道作业都设一定的优先级别，优先级别可以由用户定义，也可以由系统设定或计算确定。调度时挑选优先级高者，优先级相同时，按进程进入就绪态队列或作业进入后备态队列的先后顺序。

3. 分时和优先级相结合调度

这种策略主要用于具有分时操作的系统中，有些多道批处理系统也引入处理机分时的机制，或用于会话与批处理两种作业共存的系统，即通用操作系统中。

4. 综合考虑资源要求调度

这种策略要求综合考虑每一道作业对资源的要求。在多级调度中，内存因素也是作业调度至关重要的因素。此外，还要考虑作业对设备的要求。例如，有的系统设计调度算法要考虑用磁带与不用磁带的作业搭配运行。在作业调度时，把使用磁带机的作业和不使用磁带机的作业相间挑选。这样，能在不使用磁带机作业执行时，预先通知操作员将下一批要用的磁带装上，省去等待装带的时间。

还有的系统把作业对 CPU 时间的要求、存储空间的要求和输出行数从调度角度上建立某种等效关系，都折算为对 CPU 的时间，作为其优先级计算的依据。

以上讨论了作业调度的常用算法，有的算法也适用于进程调度。

3.4.3　优先级调度算法

优先级调度算法也叫做优先权调度算法，可作为作业调度或进程调度策略。首先，系统或用户按某种原则，为作业或进程指定一个优先级。在进程或作业进行调度时，调度程序根据优先级进行调度。这种算法可用于批处理系统，也可用于实时系统。

1. 优先级调度算法的类型

根据调度程序是否可以在一个作业或进程运行过程中抢占，优先级调度算法又分为非抢占式优先级调度算法和抢占式优先级调度算法。

（1）非抢占式优先级调度算法。若进程调度采用非抢占式优先级调度算法，系统在就绪队列中选择一个优先级最高的进程，分配给处理机后，该进程便一直执行下去，直至完成；只有因发生其他事件，使该进程进入阻塞状态，该进程才放弃处理机，系统在就绪态进程中，再找另一个优先级最高的进程，将处理机分配给它。这种调度算法主要用于批处理系统中，也可用于某些对实时性要求不严的实时系统中。

（2）抢占式优先级调度算法。若进程调度采用抢占式优先级调度算法，系统在就绪队列中选择一个优先级最高的进程，分配给处理机，使之执行。在其执行期间，如果又出现了另一个优先级更高的进程，进程调度程序就立即停止当前进程的执行，重新将处理机分配给新到的优先级更高的进程。因此，采用这种调度算法时，每当系统中出现一个新的就绪进程 i 时，就将其优先级 P_i 与正在执行的进程 j 的优先级 P_j 进行比较，如果 $P_i \leqslant P_j$，原进程 P_j 便继续执行；如果 $P_i > P_j$，则立即停止 P_j 的执行，做进程切换，使 i 进程投入执行。显然，这种抢占式优先权调度算法能更好地满足紧迫作业的要求，故而常用于要求比较严格的实时系统中，以及对性能要求较高的批处理和分时系统中。

2. 优先级的类型

在优先级调度算法中，优先级用来表示作业或进程所享有的调度优先权。该算法的关键是确定进程或作业的优先级。确定优先级的方法有两类：静态优先级和动态优先级。静态方法根据作业或进程的静态特性，在作业或进程开始执行前就确定它们的优先级，一旦开始执行后就不能改变；动态算法则不同，它把作业或进程的静态特性和动态特性结合起来确定作业或进程的优先级，随着作业或进程的执行，其优先级不断发生变化。

（1）静态优先级。静态优先级是在创建进程时确定的，且在进程的整个运行期间保持不变。通常，优先级是利用某一范围内的一个整数来表示的。例如，0～7 或 0～255 中的某一整数，把该整数称为优先数。优先数具体用法有所不同，有的系统优先数大的，优先级别高，但有的系统正好相反，优先数小的优先级别高，如 UNIX 系统。

确定进程优先级的依据有如下 3 个方面：

1）进程类型。通常，系统进程的优先级高于用户进程的优先级。对于系统进程，也可以根据其所要完成的功能划分为不同的类型。例如，调度进程、I/O 进程、中断处理进程、存储管理进程等。这些进程还可进一步划分成不同类型和赋予不同的优先级。例如，在操作系统中，对键盘中断的处理优先级和对电源掉电中断的处理优先级是不同的。

2）进程对资源的需求。例如，根据估计所需处理机的时间、内存需求、I/O 设备的类型及数量等，来确定作业的优先级。

3）用户要求。用户根据作业的紧急程度确定一个适当的优先级。当用户将自己的作业赋一个较高的优先级时，系统对该用户收取较高的费用。

通常，将作业的静态优先级作为它所属进程的优先级。

（2）动态优先级。动态优先级是指在创建进程时所赋予的优先级，是随进程的推进

或等待时间的增加而改变的，以便获得更好的调度性能。例如，我们可以规定，在就绪队列中的进程，随其等待时间的增长，其优先权也越来越高；处在运行状态的进程，它的优先级越来越低。若所有进程都具有相同的优先权初值，则最先进入就绪队列的进程，将因其动态优先权变得最高而优先获得处理机，这也就是 FCFS 算法。若所有的就绪进程具有各不相同的优先权初值，那么，对于优先权初值低的进程，在等待了足够的时间后，其优先权便可能升为最高，从而获得处理机。高响应比优先算法实际上就属于动态优先级调度算法。

例 3：有一个具有两道作业的批处理系统，作业调度采用短作业优先的调度算法，进程调度采用以优先数为基础的抢占式调度算法，假设优先数小的优先级别高。如表 3-3 所示为作业序列。

（1）列出所有作业进入内存的时间及作业结束时间。

（2）计算平均周转时间。

表 3-3　例 3 表

作业名	到达时间	估计运行时间	优先数
A	10:00	40 分钟	5
B	10:20	30 分钟	3
C	10:30	50 分钟	4
D	10:50	20 分钟	6

解：两道作业的批处理系统是指内存中能同时容纳两道作业，即系统最多能有两道作业并发执行。作业只有被作业调度程序选中，进入内存后，才能够进行进程调度。

10:00 时，A 作业到达。因系统的后备作业队列中没有其他作业，进程就绪队列中也没有进程，故作业调度程序将作业 A 调入内存并将它排到就绪队列上，进程调度程序选中它并运行。

10:20 时，B 作业到达。因系统的后备作业队列中没有其他作业，所以该作业被作业调度程序选中，加载到内存。作业 B 的优先级高于作业 A 的优先级，进程调度程序暂停作业 A 的进程运行，将作业 A 放入就绪队列，调度作业 B 运行。此时，系统中已有两道作业。作业 A 已运行 20 分钟，还需运行 20 分钟。

10:30 时，作业 C 到达。因系统已有两道作业，所以作业 C 只能在后备作业队列中等待作业调度。此时，作业 B 已运行了 10 分钟，还需运行 20 分钟才能完成。

10:50 时，作业 B 运行 30 分钟后结束运行，作业 D 到达。作业后备队列中有 C、D 两道作业，按照短作业优先的作业调度算法，作业 D 被作业调度程序选中，作业 C 仍在后备队列中等待。在内存中，作业 A 的优先级高于作业 D，因此作业 A 先运行。需 20 分钟才能完成。

11:10 时，作业 A 运行结束。系统中只有作业 D 在内存中运行，作业调度程序将作业 C 装入内存运行，因作业 C 的优先级高于作业 D，进程调度程序选中作业 C 运行，作业 D 等待。

12:00 时，作业 C 运行 50 分钟后结束运行，进程调度程序选中作业 D 运行。

12:20 时，作业 D 运行 20 分钟后结束运行。

总结以上分析，可知：

（1）各作业进入时间和结束时间如下：

表 3-4 作业时间

作业名	进入时间	结束时间
A	10:00	11:10
B	10:20	10:50
C	11:10	12:00
D	10:50	12:20

（2）根据周转时间=完成时间-提交时间，可得各作业的周转时间为：

T_A=11:10-10:00=70（分钟）

T_B=10:50-10:20=30（分钟）

T_C=12:00-10:30=90（分钟）

T_D=12:20-10:50=90（分钟）

平均周转时间＝（T_A+T_B+T_C+T_D）/4=70（分钟）

3.4.4 轮转调度算法

1. 时间片轮转法

时间片轮转法（RR 算法）主要用于分时系统中的进程调度。轮转调度的实现原理为系统把所有就绪进程按先入先出的原则排成一个队列，新来的进程加到就绪队列末尾，每当执行进程调度时，就绪队列的队首进程总是先被调度程序选中，在 CPU 上运行一个时间片的时间。时间片是一个小的时间单位，通常为 10～100ms 数量级。当进程用完分给它的时间片后，系统的计时器发出时钟中断，调度程序进行调度，停止该进程的运行，并把它放入就绪队列的末尾；随后，进行进程切换，把 CPU 分给就绪队列的队首进程，同样让它运行一个时间片，如此往复。轮转法的原理如图 3-4 所示。

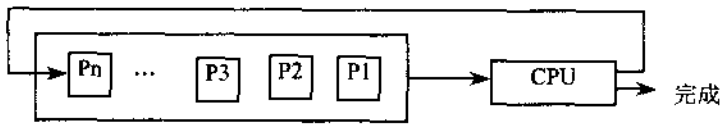


图 3-4 时间片轮转法原理图

例 4：考虑下述 4 个进程 A、B、C、D 的执行情况。设它们依次进入就绪队列，但前后时间忽略不计。4 个进程分别需要运行 12、5、3、6 个时间单位，如表 3-5（a）所示。计算当时间片分别为 q=1、q=4 时各进程的开始运行时间、完成时间、周转时间及带权周转时间。

解：通过分析和计算可以得到如表 3-5（b）所示的结果。

表 3-5　例 4 的表格及算法比较

（a）

进程名	到达时间	运行时间
A	0	12
B	0	5
C	0	3
D	0	6

（b）　当 q=1、q=4 时 RR 调度算法的比较

进程名 到达时间		到达 时间	运行 时间	开始 时间	完成 时间	周转 时间	带权周 转时间
时间片 q=1	A	0	12	0	26	26	2.17
	B	0	5	1	17	17	3.4
	C	0	3	2	11	11	3.67
	D	0	6	3	20	20	3.33
	平均周转时间 T=18.5　　平均带权周转时间 $\overline{W}$ =3.14						
时间片 q=4	A	0	12	0	26	26	2.17
	B	0	5	4	20	20	4
	C	0	3	8	11	11	3.67
	D	0	6	12	22	22	3.67
	平均周转时间 T=19.75　　平均带权周转时间 $\overline{W}$ =3.38						

上面的比较可以看出，时间片的大小对轮转法的性能有较大影响。如果时间片足够长，使每个进程都能在这段时间内运行完毕，那么时间片轮转法就退化为先来先服务算法，显然，对用户的响应时间必然大大加长。但是，如果时间片太短，那么 CPU 在进程间的切换工作就非常频繁，从而导致系统开销增加，因为每个时间片末尾都产生时钟中断，进行进程切换，这个工作需要系统开销。

时间片的长短通常由以下几个因素确定。

（1）系统的响应时间。在进程数目一定时，时间片的长短正比于系统对响应时间的要求。

（2）就绪队列中进程的数目。当系统要求的响应时间一定时，就绪队列中的进程数越多，时间片应越小。

（3）进程的转换时间。若进程的转换时间为 t，时间片为 q，为保证系统开销不大于某个标准，应使比值 t/q 不大于某一数值，如 1/10。

（4）CPU 运行指令速度。CPU 运行速度快，时间片可以短些；反之，则应取得长些。

2. 多级队列调度算法

该调度算法的主要思想是将系统中的就绪进程，按其优先级高低不同，分为两级或多级队列。进程调度时，首先从高优先级就绪队列中挑选进程，只有高一级队列为空时，才从低优先级就绪队列中选取。每个队列中可按先来先服务（FCFS）方法或轮转法进行调度。在按轮转法调度时，每个就绪队列的时间片 q 值大小可以不同。一般说来，优先级高的就绪队列，其 q 值可设较小值。

进程的优先级可以事先给定。例如，在通用操作系统中既有终端的分时作业，又有批处理作业。可以将终端上的分时用户进程定为高优先级，而把非终端用户进程定为低优先级。有的系统把使用 I/O 设备频繁者进程定为高优先级或中优先级，有的系统设更多的级，如系统作业、交互型作业、编辑型作业、批处理型作业、学生型作业等，这样，系统中就设置了多个级别的就绪队列，并赋予它们不同的优先级。也可以对每个队列采用不同的调度算法，如对系统作业可采用最高优先级优先调度算法；对交互性作业采用轮转法；对批处理型作业队列采用 FCFS 算法。仅当无系统作业时，才运行交互作业；仅当无系统型作业和交互型作业时，才运行编辑型作业；学生型作业队列优先级最低，只有当系统中无其他类型作业时，才运行学生型作业。

3. 多级反馈队列调度算法

多级反馈队列调度算法的实现思想是在多级队列法的基础上，加进“反馈”措施。

（1）系统中设置多个就绪队列，各个队列具有不同的优先级：第一个队列的优先级最高，第二个队列次之，以下各个队列的优先级逐个降低。

（2）各就绪队列中进程的运行时间片不同，高优先级队列的时间片小，低优先级队列的时间片大，如从高到低依次加倍。

（3）一个新进程进入系统后，首先被放入第一队列的末尾，各队列按 FCFS 方式排队，每个进程运行一个时间片；如某个进程在一次运行时没有完成工作，则把它转到下一级队列的末尾。

（4）系统先运行第一队列中的进程，第一队列为空后，才运行第二队列中的进程，依次类推。最后一个队列（最低级）中的进程仍然采用时间片轮转的方式进行调度。

（5）当比运行进程更高级别的队列中有一个新进程时，它将抢占运行进程的处理机，而被抢占的进程回到原队列的末尾。

多级反馈队列调度算法示意如图 3-5 所示。

4. 多级反馈队列调度算法的几点说明

（1）照顾输入输出型进程是系统的宗旨。其目的在于充分利用外部设备，以及对终端交互及时地予以响应。输入输出型进程通常进入最高优先级队列，能很快得到处理机。另一方面，第一级队列的时间片的大小也应使之大于大多数 I/O 型进程产生一个输入输出所需的运行时间。这样既使输入输出型进程得到及时的处理，也避免过多进程间转接操作，减少了系统开销。

（2）照顾了较短的作业。如果作业或进程仅在第一队列中执行一个时间片即可完成，作业的周转时间就非常短。对于稍长的作业，通常也只需在第二队列和第三队列各执行一

个时间片即可完成。

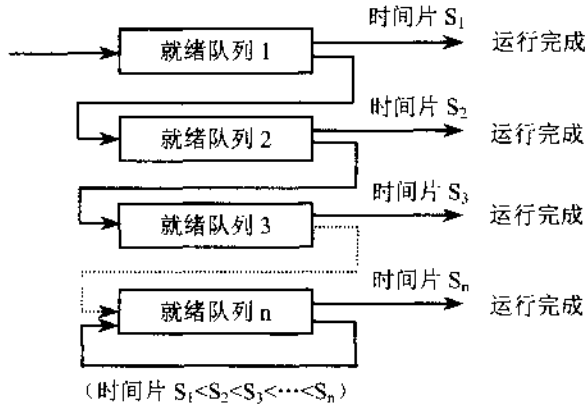


图 3-5　多级反馈队列调度算法示意图

（3）计算进程总是用尽时间片，而由最高级逐次进入低级队列，虽然运行优先级降低了，等待时间也较长，但终究得到较大的时间片来运行。同时，对于长作业，响应时间较短。

（4）在有些分时系统中，一个进程由于输入输出完成而要求重新进入就绪队列时，并不是将它放入最高级别的就绪队列，而是进入因输入输出而要求离开原来的一级就绪队列，这就需要对进程所在的就绪队列进行记录。这样做的好处是有些计算型进程，偶尔产生一次输入输出要求，输入输出完成后仍然需要很长的处理机运行时间，为减小进程的调度次数和系统开销，不让它从最高级队列逐次下降，而是直接放入原来所在的队列。

3.5　Linux 系统的调度算法

Linux 系统作业调度非常简单，或者说没有作业调度，作业一旦提交，就直接进入内存，建立相应的进程，进入下一级的调度。交换调度主要涉及系统存储管理的内容，在内存管理中讨论。Linux 系统中的内核级线程和进程在表示、管理调度方面没有差别，系统也没有专门的线程调度，采用进程调度统一处理进程和内核级线程。因此，在此主要讨论 Linux 系统中的进程调度方法。

3.5.1　Linux 系统的进程调度策略

Linux 系统的调度程序就是内核中的 schedule()函数，它的主要任务是在就绪队列 run_queue 中选出一个进程并投入运行。schedule()函数需要确定以下参数：

进程调试算法 policy；

进程过程中剩余的时间片 counter；

进程静态优先级 priority，在 Linux 2.4 版本中取消了这个变量，但是它所代表的默认值 20 作为常数还在继续使用；

实时进程的优先级 rt_priority；

用户可控制的 nice 因子。

这些参数的变量被存放在进程控制块中相应的调度成员中。

Linux 系统提供了 3 种进程调度算法，这 3 种算法可由用户通过宏定义来选择。可使用的调度策略如表 3-6 所示。

表 3-6 调度策略

调度策略标志	所代表的调度策略
#define SCHED_OTHER	普通的分时进程
#define SCHED_FIFO	先进先出的实时进程
define SCHED_RR	基于优先级的轮转算法（动态优先级调度）
#define SCHED_YIELD	不是调度策略，表示进程让出 CPU

内核把进程分为实时进程和非实时进程（普通进程）两种。实时进程获得 CPU 比普通进程优先。用户可以通过系统调用 sched_setschedule()函数改变自己的调度策略，通过系统调用 sys_setpriority()和 sys_nice()改变其静态优先级。一旦进程变为实时进程，它的子孙进程也是实时进程。

3.5.2 Linux 系统的优先级调度策略

进程的优先级是一些短整数，代表每个进程的相对获得 CPU 的权值。因此，进程的优先级越高，得到 CPU 的机会也就越大。Linux 内核又进一步把进程优先级分为静态优先级、动态优先级以及实时优先级。

1. 静态优先级

Linux 进程的静态优先级为 task_struct 结构中的一个分量 priority。

进程静态优先级 priority 是在进程建立时继承父进程的，用户可以通过系统调用 nice()或 setpriority()来设置，除此之外，priority 在一个进程整个运行期间其值保持不变。在 Linux1.0 版中，静态优先级的取值范围是 1～35，普通用户使用 nice()系统调用设置静态优先级时只能在这个范围内设置，其后的版本中新增的 setpriority()系统调用允许超级用户将一个进程的动态优先级定义为任意大的值。当然，如果将一个进程的动态优先级 counter 设得过大，系统将降级为一个单任务的操作系统。

2. 普通进程的调度策略

对于普通进程即非实时进程，Linux 的调度策略采用抢占式动态优先级调度算法。这是一种经常被采用的进程调度算法，这种算法不设进程就绪队列。进程调度子程序 schedule()总是选择动态优先级最高的进程来分配给它 CPU，如图 3-6 所示。

动态优先级是通过 task_struct 结构中的 counter 分量定义的。counter 表示在抢占式中断调度前该进程还能运行的时间，这个时间的单位是 tick，即时钟中断的发生周期。在 Linux 中，一个 tick 是 10ms，即每隔 10ms 发生一次时钟中断。因此，如果一个进程在某个时刻的动态优先级是 55，那么该进程这次使用 CPU 至少还能再连续运行 550ms，在此期间不能发生抢占式中断调度，即在此期间只能进行快中断处理，除非该进程因需等待 I/O 等原

因而主动退出运行态，否则不能被抢占式中断调度强迫退出运行态。如果某个时刻，系统中共有 4 个就绪进程，它们的优先级分别是 2、20、30、10，那么调度程序显然会选中动态优先级为 30 的那个进程来使用 CPU。从时间片的角度来看，Linux 的时间片是动态时间片，同一时刻上不同进程的时间片是不同的，同一进程在不同时刻的时间片也是不同的，Linux 的进程调度算法总是选中当前时刻时间片最大的那个进程来使用 CPU。

进程动态优先级 counter 的值的产生和变化与进程静态优先级有关。

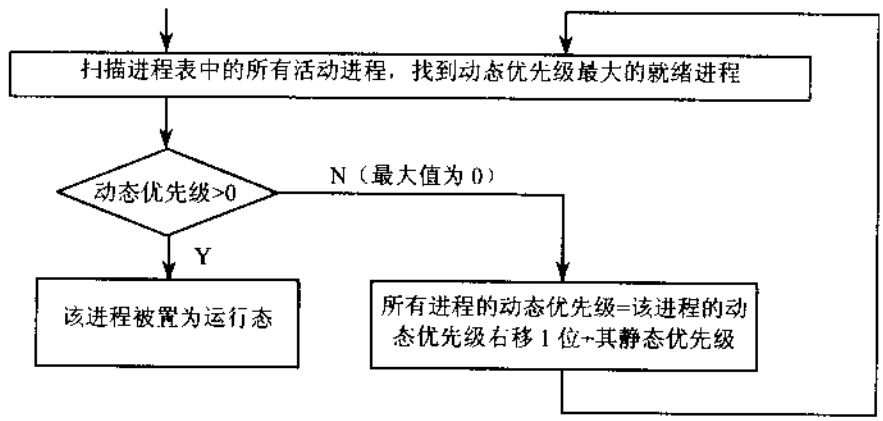


图 3-6　Linux 的进程调度过程

在进程建立时，进程动态优先级的值是从父进程（当时的值）继承的。之后，在该进程存在期间，进程动态优先级是变化的。每当时钟中断发生时，当前运行进程的 counter 值减 1，这种递减过程会在下列两种情况下停止。

（1）当 counter 递减到 0 时，进程就会在时钟中断处理过程中，被迫从运行态进入就绪态。系统中处于就绪态的进程，其进程动态优先级不一定全为零；对于那些从运行态进入就绪态的进程，其进程动态优先级一定为 0；对于那些从等待态进入就绪态的进程，其进程动态优先级通常不为 0。从运行态进入就绪态的进程，会在就绪态中一直保持动态优先级为 0，这样，不为 0 的那些进程会很快进入运行态，又很快进入等待状态或以 0 值进入就绪状态，直到系统中的所有就绪态进程的动态优先级都为 0，再在进程调度程序中被重新置为各自进程的静态优先级的值，然后按动态优先级的值依次进入运行态运行。

（2）对于处于运行态的进程，可能在 counter 还没递减到 0 时，进程就因需等待某个事件的发生而进入了等待态。处于等待态的进程，其动态优先级通常逐渐增加，也可能不变，但不可能减小。

从图 3-6 可以看出 CPU 的调度过程。请注意该过程的“N”这一分支，这一步是当所有就绪进程的动态优先级都为零时，重新计算所有进程的动态优先级，所有就绪进程的动态优先级都被设为该进程的静态优先级。这一步有 3 点需要注意：

1）并不是当所有进程的动态优先级都递减到 0 时才一起这样做；

2）所有进程都参加了重新计算，包括就绪态进程和等待态进程；

3）并不是直接将静态优先级值赋给动态优先级，而是利用一个专用的计算公式给出Linux 进程的动态优先级的计算方法。Counter 的计算公式为：

```
p->counter=(p->counter>>1)+p->priority
                    //p 为每个进程的 task 结构指针
```

对每个进程，将该进程的动态优先级的值右移一位（即除以 2 后取整）后，再加上该进程的静态优先级的值，就得到该进程的动态优先级的新值。对就绪进程，实质上仍是直接赋值（因为此时所有就绪进程的动态优先级已都为零）。但对于处于等待态的进程来说，这个公式表示等待态进程的动态优先级大于其静态优先级，但不会比静态优先级的两倍更大。

Counter 的这种变化带来了如下效果：

（1）等待态进程会有较高的优先级，而且处于等待态时间越长的进程，其进程动态优先级就越高（直至等于其静态优先级两倍，此后便不再变化，直至进入运行态），这就意味着，I/O 较多的进程（通常从运行态退出时是进入等待态）会有较高的优先级。

（2）计算较多的进程（通常从运行态退出时是进入就绪态）会有较低的优先级，因为它不仅不会像等待进程那样增加优先级，而且会在就绪态等待较长时间。

大多数操作系统都有调度队列，如 Solaris 和 Windows 2000/NT，调度算法都采用多重队列动态优先级算法。Linux 中进程动态优先级的上述变化过程在一定程度上达到了多重队列调度算法的效果。

3.5.3 实时进程的调度策略

对于实时进程，Linux 系统使用两种调度策略：先来先服务算法（FIFS）和时间片轮转法（RR）。因为实时进程对时间响应要求较高，因此为了保证实时进程能够优先于普通进程而运行，内核为实时进程增加了第三种优先级，称为实时优先级。实时优先级被保存在进程控制块 PCB 的 rt_priority 成员中，它是一个 0～99 的整数。实时进程使用静态优先级 priority 常数和动态优先级 counter 完成的功能与普通进程相同，但是 counter 不作为实时进程调度的依据，这与普通进程不同。

3.6 死锁问题

计算机系统中有很多资源，如打印机、磁带机、一个文件的索引节点等，在多道程序设计环境中，若干进程往往要共享这类资源，而且一个进程所需的资源不止一个，这样，若干进程就会相互竞争有限的资源，因得不到满足，而陷入阻塞状态。

系统发生死锁现象不仅浪费大量的系统资源，甚至导致整个系统崩溃，带来灾难性后果。所以，对死锁问题在理论上和技术上都必须给予高度重视，采取一些有效的措施，预防和避免死锁的发生，也可以采用相关的措施，检测死锁。死锁一旦发生，要解除死锁，只有这样，才能保证系统的安全与顺利执行。

3.6.1　死锁的概念

死锁是进程死锁的简称，是由 Dijkstra 于 1965 年研究银行家算法时首先提出来的，也是计算机操作系统乃至并发程序设计中非常重要但又最难处理的问题之一。掌握操作系统中对于死锁的处理方法，对于指导我们的现实生活，都会有积极的意义。

下面来看一个死锁的例子，如图 3-7 所示，假设某系统中，进程 P_1 和进程 P_2 并发执行。进程 P_1 已占有临界资源 R_1，进程 P_2 已占有临界资源 R_2。在以后运行的过程中，进程 P_1 再申请 R_2 将得不到满足，因此被加入到阻塞态队列；进程 P_2 又申请 R_1，因 R_1 被 P_1 占有，且 P_1 不能运行，因此不能释放 R_1，所以进程 P_2 也被放入阻塞态队列。这样，系统中的两个进程都因等待对方的临界资源而不能继续运行，这种现象称为死锁。该例中的临界资源 R_1 和 R_2 可能是打印机、磁带机、缓冲区等。

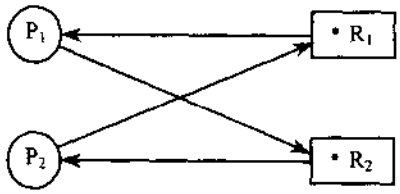


图 3-7　进程 P1、P2 陷入死锁状态

死锁是指多个进程循环等待其他进程占有的资源，因而无限期僵持下去的局面，也可以说死锁是进程之间无限期互相等待永不发生的事件。很显然，如果没有外界的作用，死锁中的各个进程将永远处于阻塞状态。

实际上，在前面的一些例子中，我们已经成功地解决过死锁问题，如生产者和消费者问题中，对于生产者进程，先做 wait(empty)，后进行 wait(mutex)；在消费者进程中，先做 wait(full)，再做 wait(mutex)，这种先后顺序也是为了避免死锁的发生。

3.6.2　产生死锁的原因及必要条件

1. *死锁产生的原因*

从上例可以看出，系统产生死锁的根本原因可归结为以下两点。

（1）各进程竞争有限的资源。系统提供的资源数量有限，远不能满足并发进程对资源的需求。可将系统中的资源分为可剥夺性资源和不可剥夺性资源。可剥夺性资源是指，某进程在获得这类资源后，该资源可被其他进程抢占，如处理机、内存。优先权高的进程可以剥夺优先权低的进程的处理机；内存区可由内存管理程序把一个进程从一个存储区移到另一个存储区，即剥夺了该进程原来占有的存储区。不可剥夺性资源是指，进程一旦占有该资源就不可抢占，不管其他进程的优先权是否高于当前进程，只能在该进程用完后自行释放，如打印机、磁带机等。

大多数情况下，引起死锁的资源竞争是指对于不可剥夺性资源的竞争。另外一种因资源竞争而死锁的资源是一些临时性资源，也称为消耗性资源，这种资源被另一进程使用很

短时间，而以后便无用的资源，如消息等。

（2）进程推进顺序不当。如图 3-7 中，两进程并发执行，占有资源并申请另一资源，这时因各进程互相等待另一进程释放临界资源而陷入死锁状态。如果此时先让进程 P_1 运行，待进程 P_1 获得它所需的两个资源 R_1 和 R_2 后，马上让 P_2 开始运行，这样就可以避免死锁的发生。以后将要讨论的银行家算法就是要找到一个合适、安全的进程推进顺序，以保证系统的正常运行。

2. 死锁产生的必要条件

通过以上对于死锁的分析，可以看出，系统如果发生死锁，一定同时存在下列 4 个条件，即死锁的必要条件。

（1）互斥条件。对于一个排它性资源，某一时刻最多只能由一个进程占有，不能同时分配给两个以上的进程。只有占有该资源的进程放弃该资源后，其他进程才能占有该资源，如打印机是临界资源，各进程必须互斥地使用。

（2）占有且申请条件。进程至少已经占有一个资源，但又申请新的资源；由于该资源已被其他进程占有，此时该进程阻塞，但是，它在等待新资源时，仍不释放已占有的资源。

（3）不可抢占条件，也称为不剥夺条件。进程所获得的资源在未使用完毕之前，资源申请者不能强行从资源占有者进程夺取该资源，只能等待占有该资源的进程释放该资源后，其他进程才可以使用。

（4）环路条件。存在一个进程等待序列{P_1，P_2，…P_n}，其中 P_1 等待进程 P_2 所占有的资源，P_2 等待进程 P_3 所需的资源，…，P_{n-1} 等待进程 P_n 所占有的资源，P_n 等待进程 P_1 所占有的资源，形成一个循环等待的环路。

上面 4 个条件是在死锁发生时同时出现的，可以利用它的逆否命题，即 4 个条件中只要有一个不满足，则死锁不会发生。这正是预防死锁所需考虑的方法。

3.6.3 解决死锁问题的基本方法

为保证系统的正常运行，应事先采取措施，预防或避免死锁的发生。在系统已出现死锁后，要及时检测到死锁并解除死锁。目前用于处理死锁问题的方法如下。

1. 死锁的预防

采取某种策略，限制并发进程对资源的请求，从而保证死锁的必要条件在系统执行的任何时间都得不到满足。

2. 死锁的避免

在分配资源时，根据资源的使用情况提前做出预测，给定一个合适、安全的进程推进顺序，从而避免死锁的发生。这种方法实现起来有一定难度，但在一些较完善的系统中常用这种方法。

3. 死锁的检测

允许系统发生死锁。系统设有专门的机构，当死锁发生时，该机构能够检测到死锁的发生，并能确定参与死锁的进程及相关资源。

4. 死锁的解除

这是与死锁检测相配套的措施，用于将进程从死锁状态中解脱出来。

由于操作系统的并发与共享以及随机性等特点，通过预防和避免死锁的手段达到排除死锁的目的十分困难，需要相当大的系统开销，对资源的利用也不够充分。死锁的检测与解除则相反，不必花费多少执行时间就能发现死锁并从死锁中恢复出来。因此，实际操作系统很多都采用了后两种方法。

3.7　死锁的预防与避免

在系统中采用各种方法都是为了防止死锁的发生，它们在实现上也分别采用了不同的原理：死锁的预防是根据死锁的 4 个必要条件，即只要有一个条件不满足，则死锁不发生；死锁的避免，是给系统中并发执行的进程找到一个安全的推进顺序。

3.7.1　死锁的预防

死锁的必要条件中，第一个条件是“互斥条件”，对于可分配的资源要互斥使用，这是由资源的固有特性决定的，不可改变。因此，只有通过打破后三个条件，使它们中的一条不成立，来达到预防死锁的目的。

1. 摒弃占有且申请条件

可以采用资源的静态预分配策略或释放已占资源策略。

（1）资源的静态预分配。在进程运行以前，一次性地向系统申请它所需的全部资源。如果某个进程所需的全部资源得不到满足，则不分配任何资源，此进程暂不执行。只有当系统能够满足当前进程的全部资源的需求时，才一次性地将所申请的资源全部分配给该进程。这种方法存在一些缺点：

1）一般情况下，一个进程在执行期前不可能知道它所需的全部资源，因为进程执行时是动态的。

2）资源的利用率低。无论所分配资源何时用到，一个进程只有在占有所需全部资源后才能执行。

3）降低了进程的并发性，因此系统的效率不高。由于资源有限，能分配到全部资源的进程数量必然减少。

（2）释放已占资源策略。仅当进程没有占用资源时才允许它去申请资源。如果进程已经占用了某些资源而再申请资源，应先归还所占用的资源后再申请新资源。例如，进程对存放在磁带机上的文件进行处理，然后将处理结果打印输出。进程可以先申请磁带机，得到磁带机后进程就可以开始执行，启动磁带机，读出文件后进行处理，把处理后的文件保存在自己的工作区。然后，释放磁带机，再申请打印机。得到打印机后，就可把处理好的文件打印输出。

这种方法仍会使一些进程处于等待资源状态。进程所申请的资源可能已被其他进程占用，只能等待占用者进程释放资源后，系统才可能分配给申请进程。

2. 摒弃不可抢占条件

我们采取的策略是隐式抢占。约定如果一个进程已经占有了某些资源又要申请另外的资源，而被申请的资源不满足时，该进程必须等待，同时释放已占有的资源，以后再进行申请。它所释放的资源可以重新被分配给其他进程，这就相当于该进程占有的资源被隐式地抢占了，这种预防死锁的方法实现起来较为困难。

3. 摒弃环路条件

实行资源的有序分配策略，即把资源事先分类编号，按序分配，使进程在申请、占有时不会形成环路。例如，令输入机的序号为 1、打印机的序号为 2、磁带机的序号为 3、磁盘的序号为 4 等。所有进程对资源的请求必须严格按照资源序号递增的次序提出，这样，在资源分配图中，保证不再存在环路。

这些预防死锁的策略与前面两种策略相比，其资源利用率和系统吞吐量都有较为明显的改善。但也存在缺点，如各类资源所分配的序号必须相对稳定，这就限制了新类型设备的增加；考虑到大多数作业在实际使用资源时的顺序，但也经常会出现作业使用各类资源的顺序，与系统规定的顺序不同，造成对资源的浪费；为方便用户，系统对用户在编程时所施加的限制条件应尽量多，然而这种按规定次序申请的方法必然会限制用户自然、简单地编程。

3.7.2 死锁的避免

以上讲到的死锁的预防是排除死锁的静态策略，它使产生死锁的 4 个必要条件不能同时具备，从而对进程申请资源的活动加以限制，以保证死锁不会发生。死锁的避免是一种排除死锁的动态策略，不限制进程有关申请资源的命令，而是对进程所发出的每一个申请资源的活动加以动态地检查，并根据检查结果决定是否进行资源分配，即在资源分配过程中预测是否会出现死锁，如不会死锁，则分配资源；若有发生死锁的可能，则加以避免。这种方法的关键是确保资源分配的安全性。

1. 安全状态

由于避免死锁的策略允许进程动态地申请资源，系统需提供某种方法在进行资源分配之前，先计算资源分配的安全性，若此次分配不会导致系统进入死锁状态，则将资源分配给进程，否则进程等待。

安全状态是指系统中的所有进程能够按照某种次序分配资源，并且依次运行完毕，进程序列$\{P_1,P_2,\cdots,P_n\}$就是安全序列。如果存在这样一个安全序列，则系统是安全的，称此时系统处于安全状态。如果系统不存在这样一个序列，则称系统是不安全的。

例 5：有 3 个客户 C1、C2、C3 向银行家贷款。该银行家的资金总额为 10 个资金单位，其中客户 C1 要借 9 个资金单位，客户 C2 要借 3 个资金单位，客户 C3 要借 8 个资金单位，总计 20 个资金单位。若 T_0时刻，客户占用和还需资源的状态如表 3-7 所示，银行家应如何分配资金？

解：银行家手里剩余的资源单位数是 10-(2+2+4)=2。此时银行家只有将资金分配给 C2 才能最终收回贷款。然后，再将资金依次分配给 C3 和 C1，这样银行家能最终收回所有贷款。这里就存在一个安全序列{C2，C3，C1}，此时系统是安全的。

表 3-7　例 5 表

客户	已分配资源	还需申请资源
C1	2	7
C2	2	1
C3	4	4

2. 由安全状态向不安全状态的转化

如果不按照安全序列的顺序分配资源，系统可能由安全状态进入不安全状态。例如，若在 T_0 时刻以后，C3 又申请到 2 个资金单位，则系统进入不安全状态。因为，此时无法再找到一个安全序列，结果造成系统产生死锁。由此可见，当 C3 申请资源时，尽管当时系统中还有可用资源，但却不能分配给它，必须让它等待。按银行家的术语说，某客户若无偿还能力时，就不要贷款给它。

3. 银行家算法

最具有代表性的避免死锁的算法是 Dijkstra 的银行家算法，由于该算法可能用于银行现金贷款而得名。一个银行家把他的固定资金贷给若干顾客，只要不出现一个顾客借走所有资金后还不够，银行家的资金应是安全的。银行家需要一个算法保证借出去的资金在有限时间内可以收回。

假定顾客分成若干次进行贷款，并在第一次贷款时说明他的最大借款额。具体算法如下：

（1）顾客的贷款操作依次顺序进行，直到全部操作完成。

（2）银行家对当前顾客的贷款操作进行判断，以确定其安全性，看能否支持顾客贷款，即该客户能否运行完成。

（3）安全时，贷款；否则，暂不贷款。

3.8　利用银行家算法避免死锁

3.8.1　银行家算法中的数据结构

1. 可利用资源向量 Available

可利用资源向量也称为空闲向量，是一个含有 m 个元素的数组。每一个元素代表一类可利用的资源数目，其初始值是系统中所配置的该类全部可用资源的数目，其数值随该类资源的分配和回收而动态地改变。如果 Available[j]=K，则表示系统中现有 R_j 类资源 K 个。

2. 最大需求矩阵 Max

最大需求矩阵是一个 n×m 的矩阵，它定义了系统中 n 个进程中的每一个进程对 m 类资源的最大需求。如果 Max[i,j]=K，则表示第 i 个进程需要 R_j 类资源的最大数目为 K。

3. 分配矩阵 Allocation

分配矩阵也叫做占有矩阵，是一个 n×m 的矩阵，它定义了系统中每一进程已占有的每一类资源数。如果 Allocation[i,j]=K，则表示第 i 个进程当前已分得 R_j 类资源的数目为 K。

4. 需求矩阵 Need

需求矩阵也叫做申请矩阵，是一个 n×m 的矩阵，用以表示每一个进程尚需的各类资源数。如果 Need[i,j]=K，则表示第 i 个进程还需要 R_j 类资源 K 个才能完成任务。

显然，前三个矩阵之间存在如下关系：

Need[i,j]=Max[i,j]-Allocation[i,j]

3.8.2 银行家算法的实现

1. 进程申请资源的情况

设 $Request_i$ 是进程 P_i 的请求向量，如果 $Request_i[j]=K$，表示进程 P_i 需要 R_j 类资源的个数为 K。$Request_i$ 与 $Need_i$ 的关系可能为以下 3 种情况：

（1）$Request_i>Need[i]$。这种情况表示该进程的资源需求已超过它所宣布的最大值，因此认为出错。

（2）$Request_i=Need[i]$。这种情况表示该进程现在对它所需的全部资源一次申请完成。

（3）$Request_i<Need[i]$。这种情况表示该进程现在对它所需资源再进行部分的申请，剩余的资源以后可再次申请。

2. 银行家算法的描述

当进程 P_i 发出资源请求后，系统按下述步骤进行检查：

（1）如果 $Request_i \leq Need[i]$，便转向步骤（2）；否则显示出错，因为它所需的资源数已超过它事先要求的最大值。

（2）如果 $Request_i \leq Available$，便转向步骤（3）；否则，表示尚无足够资源，P_i 须等待。

（3）假设系统将资源分配给 P_i，则需修改如下数据结构的值：

```
Available:= Available -Request_i;
Allocation[i]:=Allocation[i]+Request_i;
Need[i]:=Need[i]-Request_i;
```

（4）系统执行安全性算法，检查此次资源分配后，系统是否处于安全状态。如果是安全的，则将资源真正地分配给进程 P_i，否则，将本进程的试探分配作废，恢复原来的资源分配状态，进程 P_i 等待。

3. 安全性算法

工作向量 Work 表示在算法执行过程中，系统可提供给进程继续运行所需的各类资源数目，它含有 m 个元素，在执行安全算法开始时，初始值 Work:=Available。

完成向量 Finish 表示系统能否运行完成。它有 n 个分量，分别表示各进程是否可执行完成。若 Finish[i]:=true，表示第 i 个进程能够获得足够的资源，运行完成；若 Finish[i]:=false，表示该进程不能获得所需全部资源，不能运行完成。

设初值 Finish[i]:=false，i=0,1,2,…,n-1。当有足够资源分配给第 i 个进程时，再令 Finish[i]:=true。

安全算法的步骤：

（1）进行安全性检查。从进程集合中找到一个能满足下述条件的进程 i：

Finish[i]=false 并且 $Request_i \leq Work$；

若找到这样的进程，执行步骤（3）；若找不到这样的进程，则转步骤（4）。

（2）Work:= Work+Allocation[i];

Finish[i]:= true;

因为进程 P_i 若能执行完成，则能够释放它所占的资源。

（3）若所有进程的 Finish[i]=true 都满足，则表示系统处于安全状态，正式将资源分配给进程 P_i；否则，系统处于不安全状态，系统不能进行这次试探性分配，恢复原来的资源分配状态，让进程 P_i 等待。

如果已判定系统处于安全状态，则通过运算过程同时可以找到一个安全序列。如果系统处于不安全状态，则所有 Finish[i]:=false 的进程都参与了死锁。

3.8.3　银行家算法的应用

例 6：某系统有 A、B、C 类型的 3 种资源，在 T_0 时刻进程 P1、P2、P3、P4 对资源的占用和需求情况如表 3-8 所示，此刻系统可用资源向量为（2，1，2）。问：

表 3-8

资源 请求进程	最大需求量 max			已分配资源 allocation		
	A	B	C	A	B	C
P1	3	2	2	1	0	0
P2	6	1	3	4	1	1
P2	3	1	4	2	1	1
P4	4	2	2	0	0	2

（1）将系统中各种资源总数和进程对资源的需求数目用向量或矩阵表示出来。

（2）判定此刻系统的安全性。如果是安全的，写出安全序列，如果是不安全的，写出参与死锁的进程。

（3）如果此时 P1 和 P2 均再发出资源请求向量 Request(1,0,1)，为了保持系统安全性，应如何分配资源给这两个进程？说明所采用策略的原因。

（4）如果（3）中的请求都立刻满足后，系统此刻是否处于死锁状态？最终能否进入死锁状态？若能，说明参与死锁的进程，若不能，说明原因。

解：（1）系统资源总数向量= available+allocation

= (2,1,2)+(7,2,4) = (9,3,6)

进程对资源的需求矩阵

$$need=max-allocation=\begin{pmatrix}3&2&2\\6&1&3\\3&1&4\\4&2&2\end{pmatrix}-\begin{pmatrix}1&0&0\\4&1&1\\2&1&1\\0&0&2\end{pmatrix}=\begin{pmatrix}2&2&2\\2&0&2\\1&0&3\\4&2&0\end{pmatrix}$$

（2）采用银行家算法进行计算步骤如下：

```
work=(2,1,2)
finish=(false,false,false,false)
```

1）因为 need[2]<available，故系统可以满足 P2 对资源的请求，将资源分配给 P2 后，P2 可执行完成，然后释放它所占有的资源。因此：

```
finish[2]=true;
available=available+allocation[2]=(2,1,2)+(4,1,1)=(6,2,3)
```

2）此时，need[1]<available，故 P1 可执行完成。

```
finish[1]=true;
available=available+allocation[1]=( 6,2,3)+(1,0,0)=(7,2,3)
```

3）此时，need[3]<available，故 P3 可执行完成。

```
finish[3]=true;
available=available+allocation[3]=( 7,2,3)+(2,1,1)=(9,3,4)
```

4）此时，need[4]<available，故 P4 可执行完成。

```
finish[4]=true;
available=available+allocation[4]=( 9,3,4)+(0,0,2)=(9,3,6)
```

系统至少可以找到一个安全的执行序列，如（P2,P1,P3,P4），使各进程正常运行终结。

（3）系统不能将资源分配给进程 P1，因为虽然可利用资源还可以满足进程 P1 现在的需求，但是一旦分配给进程 P1 后，就找不到一个安全执行的序列保证各进程能够正常运行终结，所以进程 P1 应该进入阻塞状态。

（4）系统满足进程 P1 和 P2 的请求后，没有立即进入死锁状态，因为这时所有进程没有提出新的资源申请，全部进程均没有因资源没有得到满足而进入阻塞状态。只有当进程提出资源申请且全部进程都进入阻塞状态时，系统才处于死锁状态，但最终会进入死锁状态。

例 7：某系统有同类资源 m 个，n 个并发进程可共享该类临界资源。求每个进程最多可申请多少个该类临界资源，保证系统一定不发生死锁。

分析：要使系统不发生死锁，则每个进程能获得的资源数最大值应是让每个进程得到最大需求数减 1，另外还有一个空闲资源。这样，就有一个进程可以申请到所需资源，执行完成后，再释放资源，依次类推，最终整个系统中所有的进程都能执行完成。

解：设每个进程最多申请该类资源的最大量为 x。

每个进程最多申请 x 个资源，则 n 个进程最多同时申请的该类临界资源数为 n×x。

为保证系统不发生死锁，应满足下列不等式：

$$n(x-1)+1 \leqslant m \qquad (*)$$

这是因为进程最多申请 x 个资源，最坏的情况是每个进程都已得到了(x-1)个资源，现均申请最后一个资源。只要系统至少还有一个资源就可使其中一个或几个进程得到所需的全部资源，在它们执行结束后归还的资源可供其他进程使用，因而不可能发生死锁。

解不等式（*），可得：

$$x \leqslant 1+[(m-1)/n]$$

即 x 的最大值为 1+[(m-1)/n]。因而，当每个进程申请资源的最大数值为 1+[(m-1)/n]时，系统肯定不会发生死锁。

3.9　死锁的检测与解除

死锁的预防和避免都是对资源的分配加以限制，操作系统解决死锁问题的另一条途径是死锁的检测方法。死锁检测方法与死锁预防和避免策略不同，这种方法对资源的分配不加限制，只要有剩余的资源，就可把资源分配给申请的进程，允许系统有死锁发生，这样做的结果可能会造成死锁，关键是当死锁发生时系统能够尽快检测到，以便及时解除死锁，使系统恢复正常运行。因此，采用这种方法必须解决 3 个问题，一是何时检测死锁的发生，二是如何判断系统是否出现了死锁，三是当发现死锁发生时如何解除死锁。

3.9.1　死锁检测的时机

由于死锁检测算法允许系统发生死锁，因此，最好的情况是一旦死锁产生，就能立即检测到，也就是说死锁发现得越早越好。

死锁一般是在资源分配时发生，所以将死锁的检测时机定在有资源请求时是比较合理的，但是采用这种方法检测的次数过于频繁，若没有死锁，将占用 CPU 非常大的时间开销。

另一种方法是周期性地检测，即每隔一定时间检测一次，或者根据 CPU 的使用效率去检测，先为 CPU 的使用率设定一个最低的阈值，每当发现 CPU 使用率降到该阈值以下时就启动检测程序，以减少由于死锁造成 CPU 的无谓操作。

3.9.2　死锁的检测

1. 利用资源分配图

系统对资源的分配情况可以用有向图加以描述，该图由结对组成 G=(V,E)。其中，V 是顶点的集合，E 是有向边的集合，顶点集合可分为 $P=\{P_1, P_2, \cdots P_n\}$，由系统中全部进程组成；$R=\{r_1,r_2,r_3\}$，由系统中的全部资源组成。

由边组成的集合 E 中，每一个元素都是一个有序结对(p_i,r_j)或(r_j,p_i)。其中，p_i 是 P 中的一个进程($p_i \in P$)，r_j 是 R 中的资源类型($r_j \in R$)。如果$(p_i,r_j) \in E$，则存在一条从进程 p_i 指向资源 r_j 的有向边，进程 p_i 申请一个 r_j 资源单位。当前 p_i 在等待资源。如果$(p_j,r_j) \in E$，则有向边从资源 r_j 指向进程 p_i，表示有一个 r_j 资源分配给进程 p_i。边(p_i,r_j)称为申请边，而(r_j,p_i)称为赋给边。在资源分配图中，用圆圈表示每个进程，用方框表示各种资源的类型，方框中圆点的数量表示该类资源的个数。当然，申请边只能指向方框，而赋给边必须指向方框中的一个圆点。

如图 3-8 所示的资源分配图表示 P_1 申请临界资源 R，同时 R 已被进程 P_2 占有。临界资源 R 类中只有一个资源。

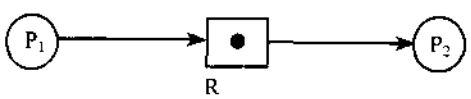


图 3-8　资源分配图

2. 资源分配图的简化

利用资源分配图进行死锁检测的目的是为了决定当前状态是否发生死锁。如果满足下列条件，那么一个连续、可重被利用的资源图就能够通过进程 P 化简：

（1）进程没有被阻塞。

（2）进程没有请求边。

（3）有分配边指向 P。

通过消除所有指向 P 的分配边，可以化简资源分配图。如果一个资源分配图不能通过任何一个进程化简，那么它就是不可被简化的；如果有一个化简序列，导致图中没有任何种类的边，那么它就是可完全简化的。

3. 死锁定理

通过资源分配图可以很直观地看出系统中的进程使用资源的情况。显然，如果图中不出现封闭的环路，则系统中不会存在死锁，如果系统出现由各有向边组成的环路，则是否产生死锁，还需进一步分析：如果环路可以通过化简的方式取消，则系统一定不产生死锁；如果环路通过化简的方式仍不能取消，即不能再进行简化，则系统一定会产生死锁，这就是著名的死锁定理。

某系统状态 S 为死锁状态的充分必要条件是，当且仅当 S 状态的资源分配图是不可完全简化的，即如果资源分配图中不存在环路，则系统不存在死锁；如果资源分配图中存在环路，则系统中可能产生死锁，如果不可再简化，则系统产生死锁。

例如，在图 3-7 中，存在进程与资源的环路 $P_1 \rightarrow R_2 \rightarrow P_2 \rightarrow R_1 \rightarrow P_1$，且不可再被简化，因此系统死锁。

例 8：如图 3-9、图 3-10 所示的资源分配图，试分析两系统是否发生死锁。

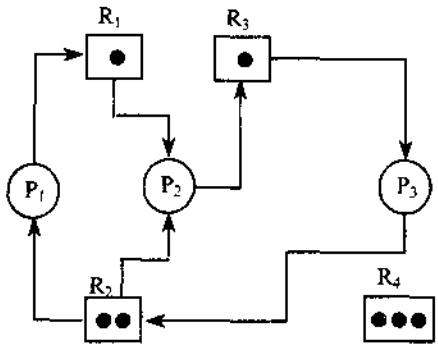


图 3-9 例 8 资源分配图

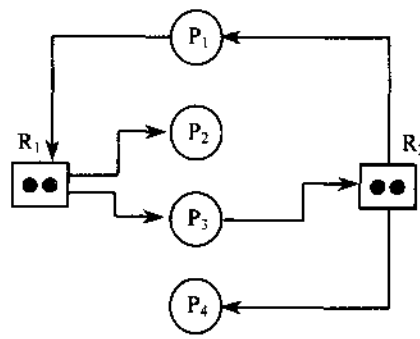


图 3-10 例 8 资源分配图

解：图 3-9 中存在两个环路 $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$ 和 $P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow \rightarrow P_2$。通过分析，不能再对该资源分配图进行简化，所以，该系统处于死锁状态，进程 P_1、P_2 和 P_3 都参与了死锁。在图 3-10 中，虽然也存在一个环路，但该资源分配图是可以简化的，因此，该系统并不死锁。

4. 死锁检测中的算法

死锁检测算法是通过对各种向量间的比较，来获得系统是否有被死锁的进程。

死锁检测算法的思想是在某时刻 t，求得系统中各类可利用资源的数目向量 available，对于系统中的一组进程$\{P_1,P_2,\cdots P_n\}$，找出那些对各类资源请求数目均小于系统现在所拥有的各类资源数目的进程，并认为这样的进程能够获得它们所需要的全部资源，并运行结束。当它们运行结束后，释放所占有的全部资源，从而使可用资源数目增加，这样的进程加入到可运行结束的进程序列 L 中（L 被初始化为空），然后对剩下的进程再作上述考查，直至再也找不到满足条件的进程。如果一组进程中有几个进程不属于序列 L，那么它们会被死锁。

当发现系统处于不安全状态时，就要执行死锁的恢复策略。

3.9.3　死锁的解除

一旦在死锁检测时发现了死锁，就要消除死锁，使系统从死锁状态中恢复过来。一般采用两种方式来解除死锁，一种是终止一个或几个进程的执行以破坏循环等待；另一种是剥夺参与死锁的进程的资源。

1. 终止进程

终止参与死锁的进程执行，系统可收回被终止进程所占用的资源，并进行再分配，以达到解除死锁的目的。

（1）终止涉及死锁的所有进程。这种方法能彻底破坏死锁的循环等待状态，但将付出很大的代价，因为有些进程可能已经运行了很长时间，由于被终止而使产生的部分结果也被删除了，在重新执行时还要再次进行计算及运行。

（2）一次终止一个进程。这种方法是每次终止一个涉及死锁的进程执行，收回它所占的资源作为可分配的资源，然后，再由死锁检测程序判断是否仍然存在死锁。若死锁依然存在，则再终止一个处于死锁中的进程，如此循环直到死锁解除。

死锁解除后，应在适当的时机让被终止的进程重新执行。当重启运行进程时应让进程从头开始执行，也有的系统在进程执行的过程中设置校验点，重新启动时让进程回退到发生死锁之前的那个校验点开始执行。设置校验点对执行时间长的进程来说是有必要的，但系统的开销较大。

2. 抢占资源

从参与死锁的一个或几个进程中抢夺资源，把释放的资源再分配给另一些参与死锁的进程，直到死锁解除。采用此方法应注意以下 3 点：

（1）抢占哪些进程的资源。希望能以最小的代价结束死锁，因此必须关注参与死锁的进程所占有的资源数，以及它们已经执行的时间等因素。

（2）被抢夺进程的恢复。如果一个进程被抢夺了资源就无法继续执行，因而应该让它返回到某个安全状态并记录有关的信息，以便重新启动该进程执行。

（3）进程的“饿死”。如果经常从同一个进程中抢占资源，则该进程总是处于资源不足的状态而不能完成，该进程就会被“饿死”。因此，一般情况下总是从已执行时间短的进程中抢夺资源，以免该现象的发生。

此外，还有进程回退策略，即让参与死锁的进程回退到没有发生死锁前的某一点处，并由此点处继续执行，以求再次执行时不再发生死锁。虽然这是个较理想的办法，但操作起来系

统开销极大，要有堆栈这样的机构记录进程的每一步变化，以便以后的回退，有时无法做到。

本章小结

本章主要介绍了计算系统的调度方法与死锁的概念。

处理机的三级调度是指高级调度是作业调度；中级调度是内外存对换；低级调度是进程调度。用户交给计算机的一个任务称为作业，作业有4个基本状态：提交状态、后备状态、运行状态、完成状态。从后备态队列中选择一个作业执行，称为作业调度。进程调度是指按照一定的算法从就绪态队列中选择一个进程运行。

调度算法的评价准则包括面向用户的准则和面向系统的准则。在面向用户的准则中，通常考虑作业的周转时间、作业的响应时间、作业的截止时间、优先权准则等；在面向系统的准则中，通常考虑系统的效率、吞吐量及各类资源的平衡利用。作业调度算法和进程调度算法有先来先服务算法、短作业优先调度算法、高响应比优先调度算法、优先级调度算法。其中，优先级调度算法包括非抢占式优先级调度算法和抢占式优先级调度算法。

死锁是指多个进程循环等待其他进程占有的资源，因而无限期地僵持下去的局面。死锁产生的原因有各进程间竞争有限的资源、进程推进顺序不当。产生死锁的必要条件有互斥条件、占有且申请条件、不可抢占条件、环路等待条件。解决死锁的基本方法有死锁的预防、死锁的避免、死锁的检测、死锁的解除。预防死锁是利用死锁的4个必要条件，只要保证一个条件不满足，死锁就不会发生；死锁的避免通常是利用银行家算法，在并发执行的进程中寻找一个安全序列，如果能找到，则系统按照安全序列的顺序分配资源，系统不会死锁。如果在进程的推进过程中，对资源的分配不加以限制，就有可能造成死锁，这时需采用死锁的检测方法，对是否陷入死锁加以检测。如果发生死锁，则采用死锁的解除策略，解除死锁。解除死锁可以利用死锁定理，即死锁的充要条件是当且仅当资源分配图是不可完全被简化的；也可利用死锁检测算法来实现。

习 题 三

3-1 处理机调度的主要目的是什么？

3-2 高级调度与低级调度的功能是什么？

3-3 处理机调度一般可分为哪3级？其中哪一级调度必不可少？为什么？

3-4 作业在其存在的过程中分为哪4种状态？

3-5 作业提交后是否马上放在内存中？为什么？

3-6 在确定调度方式和调度算法时，常用的评价准则有哪些？

3-7 什么是实时调度？与非实时调度相比，有何区别？

3-8 在批处理系统、分时系统和实时系统中，各采用哪几种进程（作业）调度算法？

3-9 在操作系统中，引起进程调度的主要因素有哪些？

3-10 假设有4道作业，它们的提交时刻及执行时间如表3-9所示：

表 3-9　习题 3-10

作业号	提交时刻	执行时间
1	10:00	2 小时
2	10:20	1 小时
3	10:40	50 分钟
4	10:50	30 分钟

计算在单道程序环境下，采用先来先服务调度算法和最短作业优先调度算法时的平均周转时间和平均带权周转时间，并指出它们的调度顺序。

3-11　什么叫死锁？死锁产生的原因和必要条件有哪些？

3-12　在解决死锁问题的几个方法中，哪种方法最易实现？哪种方法资源利用率最高？

3-13　设时间片为 1 个时间单位。现有 4 个进程，每个进程比上一个进程迟到一个时间单位，各进程要求运行时间如表 3-10 所示。系统对进程调度采用时间片轮转法。

表 3-10　习题 3-13

进程	进入时间	要求运行时间
A	0	12
B	1	5
C	2	3
D	3	1

3-14　假定某系统有 5 个进程 P_1、P_2、P_3、P_4、P_5 共享 3 类资源 A、B、C。资源类 A 共有 17 个资源，资源类 B 共有 5 个资源，资源类 C 共有 20 个资源。某一时刻，各进程对资源的需求和占用情况如表 3-11 所示。

表 3-11　习题 3-14

资源 / 请求进程	最大资源需求量			已分配资源数量		
	A	B	C	A	B	C
P_1	5	5	9	2	1	2
P_2	5	3	6	4	0	2
P_3	4	0	11	4	0	5
P_4	4	2	5	2	0	4
P_5	4	2	4	3	1	4

（1）判定系统的安全性。若安全，给出安全序列，若不安全，说明原因。

（2）在该时刻若进程 P_2 请求资源（0，3，4），是否能分配资源？为什么？

3-15　说明 Linux 系统的进程有哪几个状态？

3-16　说明 Linux 系统的进程调度算法采用什么策略？

第 4 章　存储器管理

本章学习目标

本章主要讲解存储器管理的基本方式，包括分区、页式存储管理、段式存储管理、段页式存储管理方式及虚拟存储管理方式，并以 Linux 系统为例，说明操作系统对内存的管理模式。通过对本章的学习，读者应掌握以下内容：

- 掌握存储器管理的基本概念，包括逻辑地址、物理地址、重定位等
- 掌握分区、分页式存储管理方式、分段式存储管理方式的基本思想、数据结构及重定位方法，并能灵活运用
- 掌握虚拟存储器的概念及实现原理。掌握虚拟存储器中的页面置换算法，并能灵活运用
- 理解段页式存储管理技术
- 了解 Linux 操作系统的存储管理方法

4.1　存储器管理概述

存储器是计算机系统的重要组成部分，近年来，随着计算机技术的发展，系统软件和应用软件在种类、功能及其所需存储空间等方面急剧地膨胀，虽然存储器容量也一直在不断扩大，但仍不能满足现代软件发展的需要。因此存储器仍然是一种宝贵且紧俏的资源。如何对它们施行有效的管理，不仅直接影响到存储器的利用率，而且对系统性能有重大影响。存储器管理讨论的主要对象是内存，主要讨论对内存的管理方法及有效利用。

存储管理所研究的主要内容包括 3 个方面：取（Fetch）、放（Placement）和替换（Replacement）。“取”是研究应将哪道程序（或程序的一部分）从辅存调入主存，一般有请调和预调之分。前者是按照需要来确定调入主存的程序（或程序的一部分）；后者是采用某种策略，预测并调入即将使用的某道程序（或程序的一部分）到主存。“放”是研究将“取”来的程序（或程序的一部分）按何种方式存放在主存的什么地方。“替换”是研究应将哪道程序（或程序的一部分）暂时从主存移到辅存以腾出主存空间供其他程序（或程序的一部分）占用。在这 3 个方面中，“放”是存储管理的基础。目前“放”的技术归结为两类，一类是连续的，即运行的程序必须存放在主存的一片连续空间中，如本章介绍的单一连续分配、固定分区分配、可变分区分配；另一类是不连续的，即运行的程序可以放在主存的多个不相邻的块中，如本章将介绍的页式管理、段式管理和段页式管理。

4.1.1　存储器的层次

程序是以文件的方式存储在辅存上，为了执行或存取程序和数据，需要把它们放到内存，但在具体系统中，内存容量有限，为了支持多道程序运行和大量的数据处理，就利用磁盘、磁带等容量巨大、价格便宜的存储器作为内存的后备支撑，CPU 不能直接存取外存上的信息，但内外存之间可以相互传递数据。只有将程序和数据存到内存中才能执行和访问。外存的运行速度低于内存的存取速度。

为了进一步提高运算速度和增强处理能力，可采用高速缓冲存储器来存放程序和数据。高速缓存器是由硬件寄存器构成的，因而其存取速度相当快，等同于构成寄存器的电子线路的开关速度，CPU 可以直接存取其中的信息，但成本远远高于内存的成本。因此，在一个实际系统中，缓存的容量不可能太大。

如图 4-1 所示是存储器的体系结构。

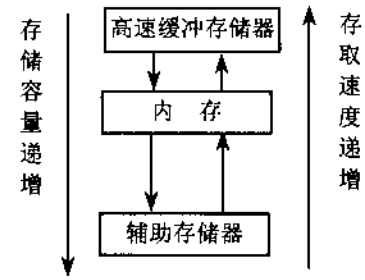


图 4-1　多级存储器体系示意图

4.1.2　用户程序的处理过程

1. 用户程序的处理过程

系统对用户程序的处理分为以下几个阶段：

（1）编译。由编译程序将用户源代码编译成若干个目标模块。

（2）链接。由链接程序将编译后形成的目标代码以及它们所需的库函数链接在一起，形成一个装入模块。

（3）装入。由装入程序将装入模块装入内存。

具体处理过程示意如图 4-2 所示。

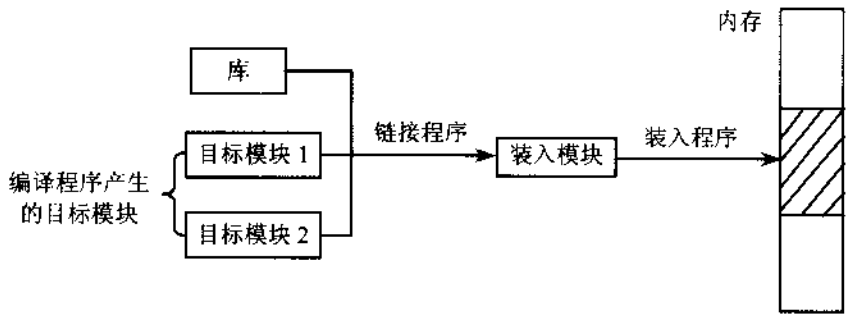


图 4-2　对用户程序的处理步骤

2. 内存管理中的几个相关概念

（1）逻辑地址是一个操作数在逻辑地址空间中的地址。逻辑地址空间是程序及数据所构成的空间，从 0 开始编址。如一个操作数的逻辑地址为 300B，则该操作数据的位置在离程序开始 300 字节处。

（2）物理地址是一个操作数加载内存后的地址，相对于内存 0 地址编址。如一个操作数的物理地址为 4MB+300B，则指该操作数在内存从 0 地址开始，做偏移 4MB+300B 所在的位置。

（3）重定位是指将一个操作数的逻辑地址转换为物理地址。

3. 目标程序装入内存的方式

程序只有装入内存后才能运行。装入方式分为绝对装入方式、可重定位装入方式和动态运行时装入方式。

（1）绝对装入方式。在编译时，如果知道程序将驻留在内存什么位置，那么编译程序将产生绝对地址的目标代码。绝对装入程序按照装入模块中的地址，将程序和数据装入内存。装入模块被装入内存后，无须对程序和数据的地址进行修改，程序中所使用的绝对地址既可以在编译或汇编中给出，也可以由程序员直接给予。一般不允许程序员给予地址，通常情况是在程序中采用符号地址，然后在编译或汇编时，将这些符号地址再转化为绝对地址。

（2）可重定位装入方式。在程序执行之前，由操作系统的重定位装入程序完成。一般用于多道程序环境中，编译程序不能预知所编译的目标模块在内存什么地址。重定位程序根据装入程序的内存起始地址，直接修改所涉及的逻辑地址，将内存的起始地址加上逻辑地址得到正确的内存地址。

如图 4-3 所示，用户程序的 2000 号单元处有一指令 load 1，3500。该指令的功能是将 3500 号单元的整数 360 取至 1 号寄存器。装入情况如图 4-3 所示，由于装入的地址是 10000，所以如果还是从 3500 号单元取数，必然是错误的。正确的方法是将指令的 3500 号单元改为 13500，即将起始地址和逻辑地址相加，得到正确的物理地址，然后从该物理地址中将其操作数 360 取出。

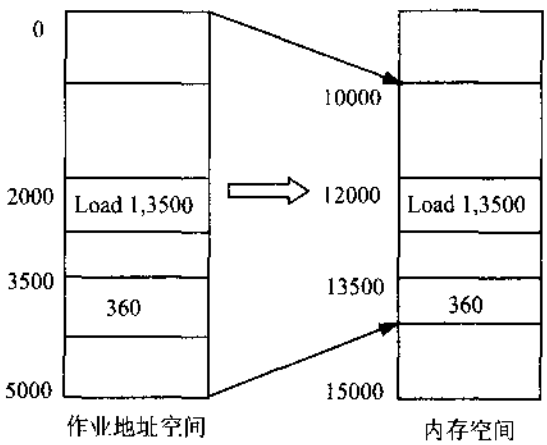


图 4-3 作业装入内存时的情况

（3）动态运行时的装入方式是在程序执行期间进行的。一般说来，这种转换有专门的硬件机构来完成，通常采用一个重定位寄存器，每次进行存储访问时，将取出的逻辑地址加上重定位寄存器的内容，形成正确的内存地址，该方法一般用于多道程序环境中。因为多道程序运行时，进程可能经常换进换出，每次换入后的位置是不相同的，在这种情况

下，静态重定位不能解决问题，需要用动态重定位，如图 4-4 所示。

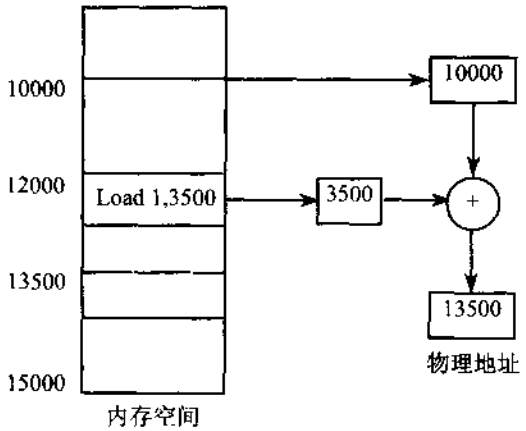


图 4-4　采用动态重定位时内存空间及地址重定位示意图

4. 目标程序链接

链接程序的功能是将经过编译或汇编后得到的一组目标模块以及它们所需的库函数装配成一个完整的装入模块。实现链接的方法有静态链接、装入时动态链接和运行时动态链接。

（1）静态链接。设编译后得到的 3 个目标模块 A、B、C，它们的长度分别为 L、M、N。程序链接示意图如图 4-5 所示。需要完成的工作是对相对地址进行修改，同时变换外部调用符号，将每个 CALL 语句改为跳转到某个相对地址，从而形成一个完整的装入模块，又称可执行文件，通常不再拆开，运行时可直接装入内存。这种事先进行链接，以后不再拆开的方式称为静态链接。

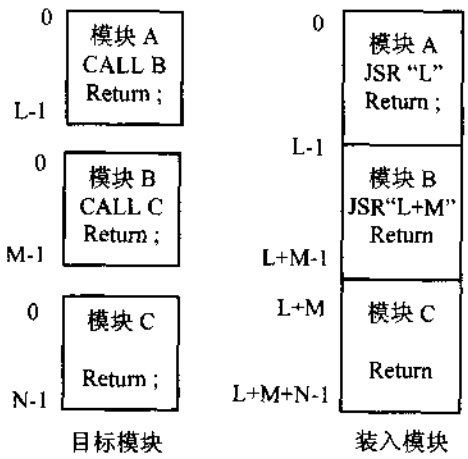


图 4-5　程序链接示意图

（2）装入时动态链接。用户源程序经编译后得到目标模块，在装入内存时边装入边链接，即在装入一个目标模块时，若发生一个外部模块调用，装入程序将去寻找相应的外部目标模块，并将它装入内存。

（3）运行时动态链接。装入时进行的链接虽然可以将整个模块装入内存的任何地方，但装入模块的结构是静态的。在程序执行期间装入模块是不可改变的，因为无法预知本次要运行哪个模块，只能将所有可能要运行的模块，在装入时全部链接在一起，使得每次执行的模块都相同，这样效率很低，因此采用运行时动态链接。在这种链接方式中，可将某些目标模块的链接推迟到执行时才进行，即在执行过程中，若发现一个被调用模块尚未装入内存时，由 OS 去找该模块，将它装入内存，并把它链接到调用模块上。

4.2 连续分配存储管理方式

连续分配是指为一个用户程序分配一个连续的内存空间，包括单道程序的连续分配和多道程序的连续分配。连续分配又称为分区分配方式，包括固定分区、动态分区和动态重定位分区 3 种。

4.2.1 单道程序的连续分配

单道程序的连续分配是一种最简单的存储分配方式，只能用于单用户、单任务的操作系统。在这种存储管理方式中，内存分为系统区和用户区两个分区，如图 4-6 所示。

1. 系统区

系统区仅供操作系统使用，一般驻留在低址部分，其中包括中断向量。中断向量是操作系统核心部分的功能模块加载内存的起始地址，该地址在系统调用时被使用。例如，MS-DOS 系统采用了这种存储管理方式。在该系统中，共 256 个中断向量，每个中断向量占 4B，因此，系统的中断向量区占 1K，该部分被分配在内存 0 地址开始的区域。

2. 用户区

操作系统所占区域以外的全部内存空间。

为了避免用户程序执行时访问了操作系统所占空间，应将用户程序的执行严格控制在用户区域，称之为存储保护，保护措施主要是由硬件实现。硬件提供界地址寄存器和越界检查机构。将操作系统所在空间的下界 a 存放在界地址寄存器中，用户程序执行时，每访问一次主存，越界检查机构便将访问主存的地址和界地址寄存器的值进行比较，若出界则报地址错，如图 4-7 所示。

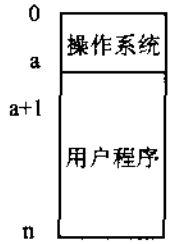


图 4-6 单一连续分区的内存结构

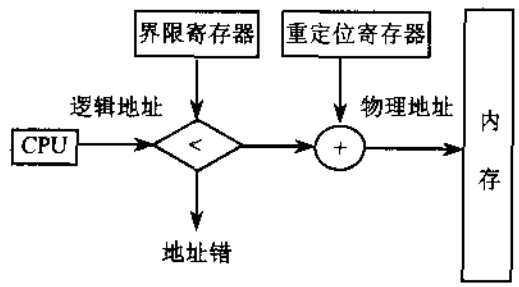


图 4-7 地址映射和地址保护

4.2.2　固定分区分配方式

固定分区管理方式比较简单，在这种管理方式下，操作系统启动时就划分好分区的大小和个数，在程序运行期间，整个系统中的分区不再变化。

例 1：设一个容量为 32k 的实际内存，分割成如图 4-8 所示各区域。

这种分区分配方式在整个系统运行期间是不变的。在这种情况下，为一个作业分配空间时，应先根据一定的分区分配策略，为作业选择一个分区，然后再进行分配。

假设有一个作业队列，如图 4-9 所示。

操作系统	10k
小作业	4k
中等作业区	6k
大作业区	12k

图 4-8　内存空间的分配

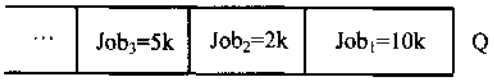


图 4-9　作业队列

显然，如果要将 3 个作业都加载内存，应将 job_1 分配在 12k 的区域，job_2 分配在 4k 的区域，job_3 分配在 6k 的区域。

一旦一个区域分配给某个作业后，它所剩余空间便不能再用，称为内存的内碎片；一个作业加载内存时，如果内存中分区的长度小于作业程序的长度，则该程序无法装入，此时称该分区为外碎片。内、外碎片之和构成了存储空间总的浪费。

若将上面的作业流改为如图 4-10 所示情况，显然 job_1 应分在 12k 的区域，剩余 4k，job_2 应分在 2k 的区域，job_3 应分在 6k 的区域。当 job_4 到达时，虽然它只需要 3k 空间，而 job_1 所在区域有 4k 的剩余，但不能把剩余的 4k 分给 job_4，于是，这 4k 的空间成了内碎片。当 job_2 完成并释放了 2k 的区域后，job_4 还是不能装入，因为 2k 区域小于 job_4 的需求量，于是，2K 的区域成了外碎片。由于一个区域最多只能放一个作业，所以区域的个数也是能并发执行的作业的最大道数，故有时也把这种管理方法称为 MFT，即道数固定的多道程序设计管理方法。

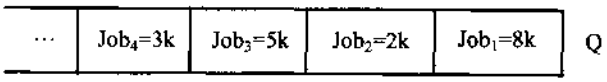


图 4-10　作业流

固定式分区管理方式的另一个问题是如何划分区域、划分几个区域、每个区域占据多大的空间，这往往要根据作业流的具体情况确定，即分别算出小作业、中作业和大作业的平均大小，然后根据其平均值划分区域。在实际系统中，为了灵活，也可以事先准备几种划分方法，操作员可根据现行作业流的不同情况选用不同的划分方法。

4.2.3　动态分区分配

可变式分区管理不是事先将内存空间一次分定，而是根据实际需要动态地分割内存。

当有些作业运行完毕并释放了其所占用的空间时，只要可能，便将那些较小的自由空间合并成较大的空间。在这种管理方法下，内存区域的个数、各区域的大小、装入内存的作业的道数都随时间而变化，所以这种管理方法也称为 MVT，即具有可变道数的多道程序设计。

动态分区分配需要解决的问题有分区分配中所用的数据结构；分区的分配算法；分区的分配与回收操作。

1. 分区分配中的数据结构

要实现分区分配，系统必须配置相应的数据结构，用来记录内存的使用情况，为分配和程序运行提供依据。常用的数据结构有空闲分区表和空闲分区链。

（1）空闲分区表结构如表 4-1 所示。

表 4-1 空闲分区说明表

序号	分区大小（KB）	分区始址（K）	状态
1	64	44	可用
2	24	132	可用
3	40	210	可用
4	30	270	可用
5	…	…	…

（2）空闲分区链。为了实现对空闲分区的分配与链接，在每个分区的起始部分设置一些用于控制分区分配的信息，以及用于链接各分区的前向指针，形成一个链表。其结构如图 4-11 所示。

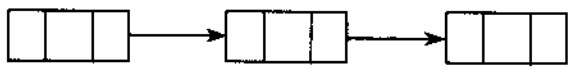


图 4-11 空闲链结构

2. 分区分配算法

为把一个新作业装入内存，应按照一定的分配算法，从空闲区表或空闲分区链中选一个分区分配给该作业，目前常用以下 3 种分配算法。

（1）首次适应算法（最先符合）。此算法可以在上述两种数据结构上实施，但通常要求自由块按始地址从低到高排序。当需分配空间时，总是从头开始查找，直到找到一个符合要求的自由块。

（2）最后适应算法（最后符合）。从内存最高地址开始查找，找到第一个满足要求的内存块，进行分配。

（3）最佳适应法。可以在上述两种数据结构上实施，但要求按自由块从小到大的顺序排序。分配从头开始查找，即从小端到大端的方向查找，直到找到第一个满足要求的自由块。显然，找到的自由块是能满足要求的最小块。

（4）最坏适应算法。数据结构和排序方法同上。当分配空间时不是从小往大查，而

是从大往小查，因此，找到的自由块是所有自由块中最大者。

评价一个算法好坏的标准有两条，一是算法本身的时间复杂度，二是此算法下的空间利用率。就算法本身来讲，复杂度由排序（按地址大小或块的大小）和查找（查找所需要的自由块）两项决定。4 个算法都要排序，这一点是相同的，所以它们的差别仅在查找上。显然，最坏适应法查找最快，它只要在内存块大的方向上考察第一个自由块，便可以得知此次是否可分配；从内存使用的角度来讲，首次适应法可以得到最快的速度。最佳适应算法具有最好的内存利用率，但实现起来查找需花费更多的时间；最后适应算法有利于利用内存的高地址区域；最先适应算法则优先分配内存的低地址区域，在实际系统中，首次适应法用得比较多。

3. 分区分配和回收

动态分区存储管理方式中，主要操作是分配和回收内存。

（1）分配内存。首先，系统利用某种内存分配算法，从空闲区表中找到所需的分区。设所请求分区的大小为 u.size，分区表中每个分区的大小可表示为 m.size。若 m.size-u.size ≤size（size 是事先规定的、不再切割的剩余分区的大小），说明多余部分太小，可不再切割，将整个分区分给请求者；否则，从该分区中划分出与请求的大小相等的内存空间并分配出去，余下的部分仍留在空闲分区链或空闲分区表中。最后，将分配的首地址返回给调用者。

（2）回收内存。回收分区的主要工作是首先检查是否有临接的空闲区，如有则合并，使之成为一个连续的空闲区，避免形成许多离散的小分区；之后，修改有关的分区描述信息。一个回收分区邻接空闲区的情况有 4 种，如图 4-12 所示。第一种情况是回收分区 r 上邻的一个空闲区，此时应合并成为一个连续的空闲区，其始地址为 r 上邻的空闲区始地址，大小为二者之和，第二种情况与 r 下面的空闲区相邻，直接合并；第三种情况是与上下空闲区相邻，将 3 个区域合并成一个连续的空闲区；第四种情况不和任何空闲区相邻，应建立一个新的空闲区，并加到自由主存队列中。

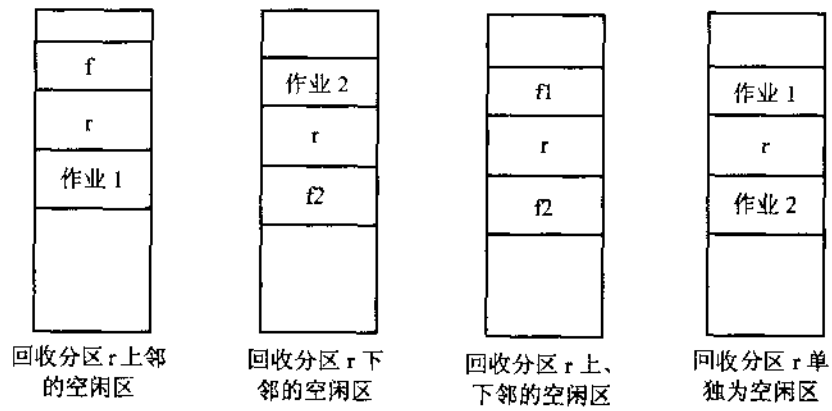
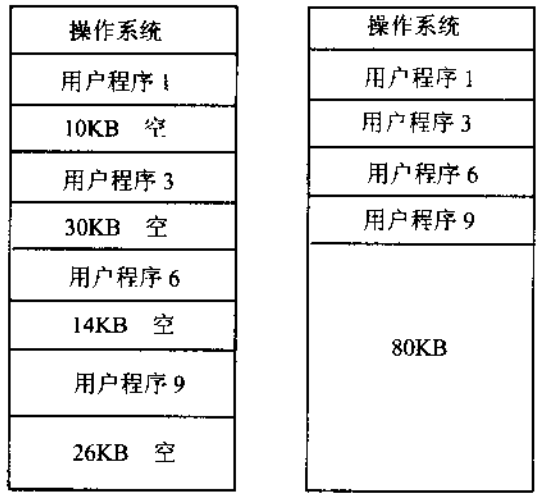


图 4-12　内存回收时的情况

4.2.4 可重定位分区

1. 紧凑

在连续分配方式中，必须把一个系统程序或用户程序装入到连续的内存空间中，如果系统中存在若干个小的分区，每个小分区都不能满足待装入程序的要求，该程序就不能装入内存，但各空闲分区之和大于要装入的程序，如图 4-13（a）所示，紧凑后如图 4-13（b）所示。



（a）紧凑前　　（b）紧凑后

图 4-13 紧凑的示意

在图 4-13（a）中，出现了 4 个不相邻的内存空间，共 80KB，如果有一个作业到达，要求申请 40KB 的内存空间。由于必须为其分配一连续空间，故此作业无法被装入。在这种情况下，要想装入作业，可采用的方法是将内存中的作业进行移动，使它们相邻，使原来许多分散的小分区可以拼接成大的分区，称为“紧凑”。通过作业平移，得到了图 4-13（b）中的 80KB 的空闲分区，作业可以装入。由于经过紧凑的用户程序在内存中的位置发生了变化，若不对程序中的数据地址进行修改变换，程序无法执行，因此必须进行重定位。

2. 动态重定位

在该方式中，将程序中的相对地址转换为物理地址的工作被推迟到程序指令执行时进行，这必须获得硬件地址变换机构的支持，即在系统增加一个重定位寄存器，用它来装入程序在内存中的起始地址。程序在执行时，真正访问的内存地址是由程序中的逻辑地址与重定位寄存器中程序加载内存的起始地址相加而形成的。地址变换过程是在程序执行期间随着对每条指令和数据访问而进行的，故称为动态重定位，如图 4-14 所示描述了动态重定位的原理。当系统对内存进行了“紧凑”，而使若干程序从内存的某处移到另一处时，不需要对程序做任何修改，只需将该程序在内存的新的起始地址置换原来起始地址即可。

3. 动态重定位分区分配算法

动态重定位分区分配算法与动态分区分配算法基本相同，差别仅在于这种分配算法中

增加了“紧凑”功能。通常，若找不到足够大的空闲区来满足用户的需要，则进行“紧凑”，然后寻找合适的内存空间。

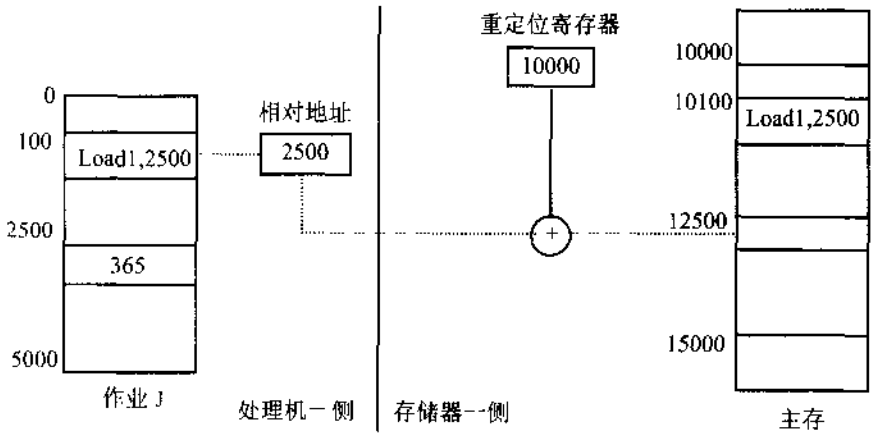


图 4-14　动态重定位示意图

4.3　分页存储管理方式

4.3.1　页式存储管理的系统应解决的问题

采用“紧凑”技术解决按区分配中存在的碎片问题，其实质是让存储器适应程序对连续性的要求。逻辑地址空间是从 0～XKB 一个连续的范围，要求主存中也有一个连续的分区用以加载。当主存现有的空闲区都小于程序的地址空间时，只有采用“紧凑”技术把碎片连接成一个大的空闲区才能满足作业的要求，但这是以花费 CPU 时间为代价换来的。这种办法只有在分配区的数目不太多，而且分配不太频繁的情况下才采用。

为了寻找解决碎片问题的新途径，人们很容易想到让程序不连续存放，例如，有一个作业要求投入运行，其程序的地址空间是 3KB，而主存当前只有两个各为 1KB 和 2KB 的空闲区，显然各空闲区的长度都分别小于该程序的逻辑空间大小，而各空闲分区的总和与程序逻辑空间大小相同，这样考虑将程序分开存放，放在不相邻的两个区域中，这正是分页的思想。在分页存储管理中，主存被分成一些大小相等的物理块，程序的地址空间被分成一些逻辑页面，逻辑页与物理块大小相同。为程序分配内存空间时，程序和数据以页为单位分配内存块，将一个逻辑页存放在一个物理块中。为了便于实现动态地址变换，通常物理块的大小为 2^n 个扇区，如 1K、2K、4K 等。

4.3.2　分页存储管理的基本方法

1. 页面和物理块

在分页存储管理中，将一个进程的逻辑地址空间分成若干个相等的片，称为页面或页。

相应地，内存空间也分成与页大小相同的若干个存储块，称为物理块或页框。为它们从 0 开始依次编号，如图 4-15 所示。为进程分配内存时，以块为单位将进程中若干页分别装入不相接的块中。由于进程的最后一页经常装不满一块，而形成不可利用的碎片称为页内碎片。每个作业最多能够形成一个页内碎片，它的长度小于 1 个页面大小。

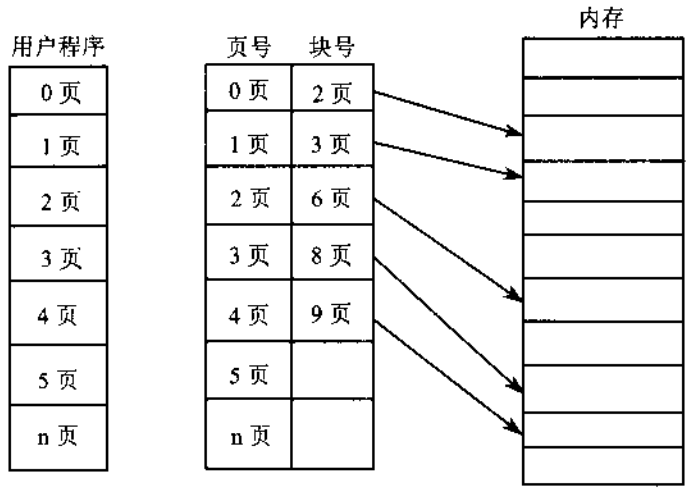


图 4-15 页表机制

2. 页表

在分页系统中，允许进程的每一页离散地存储在内存中，即分别占据一些不连续的物理块，系统应能保证进程的正确运行，即能在内存中找到每个页面所对应的物理块。系统又为每个进程建立一张页面映像表，简称页表。在进程地址空间的所有页内（0～n），依次在页表中有一页表项，其中记录了相应页在内存中对应的物理块，如图 4-15 的中间部分。可见页表的作用是实现了从页号到物理块号的地址映象。即使在简单的分页系统中，也常在页表的表项中设置一存取控制字，用于对存储块中的内容进行保护。

3. 虚地址结构

利用页表进行地址变换的方法与计算机所采用的地址结构有关，而地址结构又与所选择的页面大小有关。例如，当 CPU 给出的虚地址长度为 32 位，页面的大小为 4KB 时，内存中共有 2^{20} 个物理块。分页系统中的地址格式如图 4-16 所示。

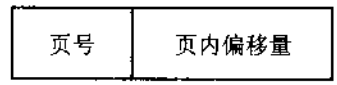


图 4-16 虚地址结构

4. 页式地址变换

例 2：如图 4-17 所示，用作业 2 的程序中一条指令的执行过程说明页式系统中的地址变换过程。设程序的地址空间中第 100 号单元处有一条指令为“mov r1,[2500]”。这条指令在主存中的实际位置为 2148 号单元（第 2 块 100 号单元），而这条指令要取的数 123 在程

序的地址空间中位于 2500 号单元（第 2 页的 452 号单元）处，在主存中存于 7620 号单元（第 7 块 452 号单元）。假设页面大小为 1KB，机器的地址长度为 16 位。

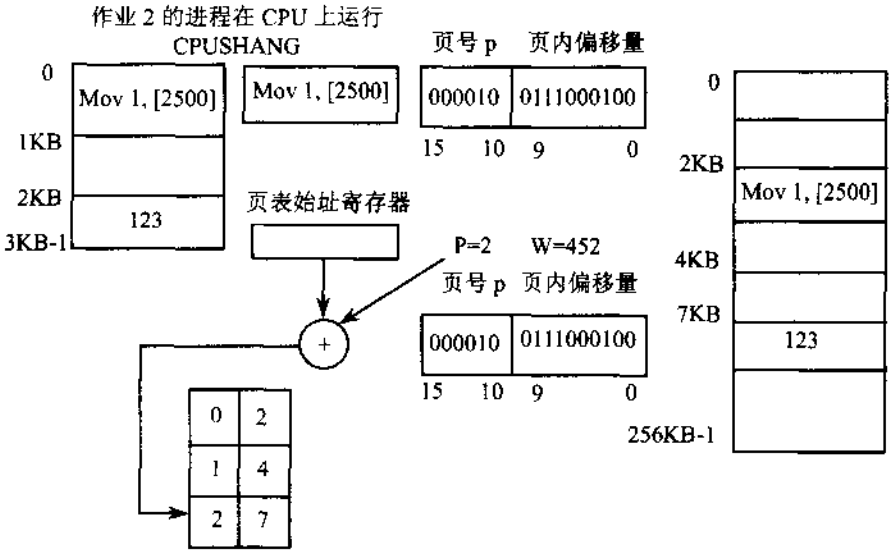


图 4-17　页式系统地址变换结构

当作业 2 的相应进程在 CPU 上运行时，操作系统负责把该作业的页表在主存中的起始地址（a）送到页表起始地址寄存器中，以便在进程运行期间进行地址变换时由它控制找到该作业的页表。

当作业 2 的程序执行到指令“mov r1,[2500]”时，CPU 给出的操作数地址为 2500，首先由分页机构自动把它分成两部分，得到页号 p=2，页内相对位移 w=452。然后，根据页表始地址寄存器指示的页表始地址，以页号为索引，找到第 2 页对应的块号 7，将块号 7 和页内位移量 w 拼接在一起，就形成了访问主存的物理地址 7620，这正是所取数 123 所在主存的实际位置。

由上述地址变换过程可知，在页式存储管理方式下，程序员编制的程序或由编译程序给出的目标程序经装配连接后形成一个连续的地址空间，其地址空间的分页是由系统自动完成的，而地址变换是通过页表自动、连续进行的，系统的这些功能对用户或程序员来说是透明的。因为在分页系统中地址变换过程主要是通过页表来实现的，因此称页表为地址变换表或地址映像表。

5. 快表

从上面介绍的地址变换过程可知，若页表全部放在主存储器内，则取一个数据（或一条指令）至少要访问两次内存，一次是访问页表，确定要取的数据或指令的物理地址；第二次是根据所取的物理地址，到内存相关地址单元取要访问的操作数。写入一个数据情况也是一样。为了提高查表的速度，减少访问内存的次数，可以考虑将页表放在一个高速缓冲存储器中。高速缓冲存储器一般是由半导体存储器实现的。现在，有的计算机系统用硬

件实现地址变换，也有一些系统将部分页表放在快速存储器中，其余部分仍放在主存中。存放页表部分内容的快速存储器称相联存储器，也称为联想寄存器，联想寄存器中存放的部分页表称为“快表”，它的格式如图 4-18 所示。这样的联想存储器一般由 8～16 个单元组成，它们用来存放正在运行进程的当前最常用的页号和它相应的块号，并具有进行查找的能力。例如，CPU 给出虚地址为(p,w)，分页机构自动把页号送入联想存储器，然后立即和其中的所有页号比较，如与某单元的页号符合，则取出该单元中的块号 b，然后用(b,w)访问存储器。这样，和通常的执行过程一样，只要访问一次内存，就可以取出指令或存取数据。如果需要查的页号和联想存储器中所有的页号不匹配，则地址变换过程还需再通过主存中的页表进行。实际上这二者是同时进行的，即一旦联想存储器中发现有所要查找的页号，就立即停止查找主存中的页表。如果地址变换是通过查找主存中的页表完成，还应把这次所查的页号和查得的块号一并放入联想存储器的空闲单元中，若无空闲单元，则通常把最先装入的那个页号淘汰掉，以腾出位置。采用联想存储器和主存中页表相结合的分页地址变换过程如图 4-18 所示。

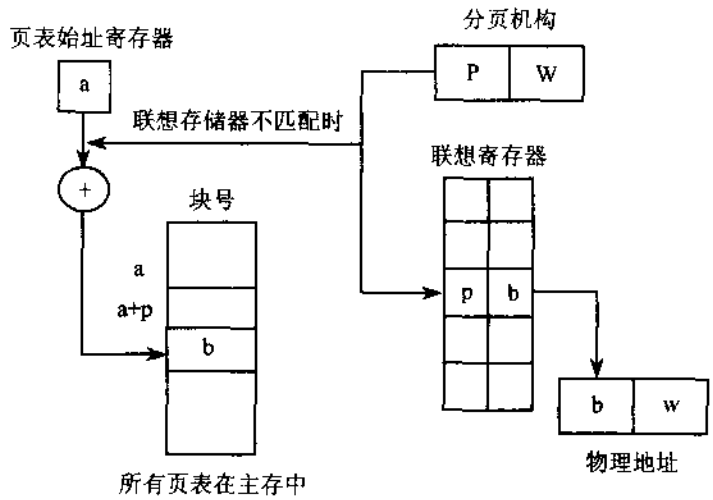


图 4-18 带有快表的分页地址变换

4.3.3 两级和多级页表

现代大多数计算机系统都支持大的逻辑地址空间（2^{32}～2^{54}）。在这样的环境下，页表就变得非常大，要占用相当大的内存空间。例如，对于一个具有 32 位逻辑地址空间的分页系统，规定页面的大小为 4KB，则每个进程页表中的页表项可达 1MB（2^{32}/4K）。假设每个页表占用 4 个字节，则仅页表就要占用 4MB 的内存空间，而且还要求是连续的。显然这是不现实的，因为页表所需的内存空间采用离散分配方式来解决，难以找到连续的较大内存空间；只将当前需要的部分页表调入内存，其余的页表项驻留在磁盘上，需要时将它们调入内存。因此，利用两级页表或多级页表来解决这一问题。

1. 两级页表

针对难以找到大的连续存储空间存放页表的问题，可利用再将页表进行分页的办法，使每个页面的大小与内存物理块的大小相同，并为它们编号。离散地将各个页面分别放在不同的物理块中，同样为每个离散的页面建立一张页表，称为外层页表。在每个页表项中记录物理块号，如图 4-19 所示。

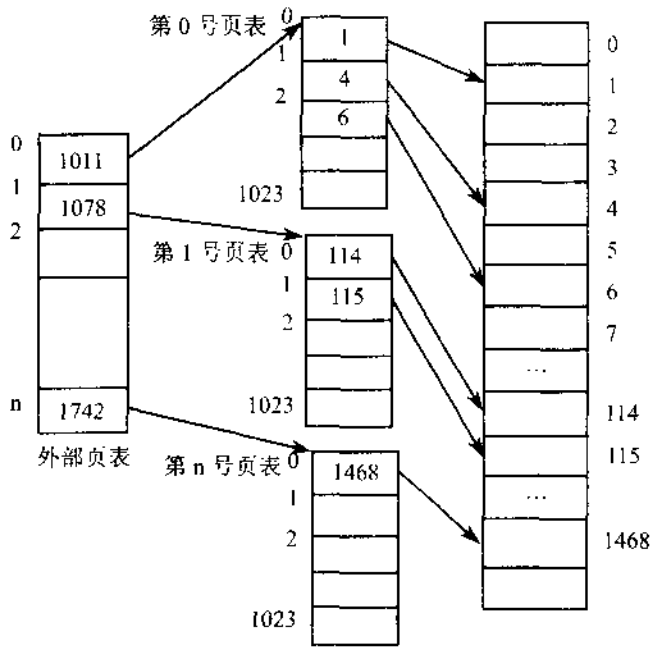


图 4-19　带有快表的分页地址变换

由图 4-19 可以看出，页表的每个表项中存放的是进程某页在内存中的物理块号，如第 0 号页存放在 1 号物理块中，1 号页存放在 4 号物理块中，而在外层页表的每个页表项中，所存放的是内层页表的首地址。如 0 号页表存放在第 1011 号物理块中。可以利用外层页表和（内层）页表实现从进程的逻辑地址到物理地址的变换。

为了实现地址变换，在地址机构中同样需要设置一个外层页表寄存器，用于存放外层页表的始地址，并利用逻辑地址的外层页号，作为外层页表的索引，从中找到指定页表分页的首地址；再利用 P2 作为指定页表分页的索引，找到指定的页表项，其中含有该页在内存中的物理块号，用该块号和页内地址 d 可构成访问内存的物理地址。如图 4-20 所示为两级页表的地址变换机构。

2. 多级页表结构

对于 32 位处理器，采用两级页表的结构是合适的，但对于 64 位处理器，两级页表是否合适，需要进一步分析。如果页面大小仍采用 4KB，那么还剩 52 位，假定仍按物理块的大小（2^{10} 位）来划分页表，则将所有剩余的 42 位用于外层页号。此时外层页表中可能有 4096G 个页表项，即使按 2^{20} 位来划分页表，每个页表分页将达 1MB，外层页表仍有 4G

个页表项，要占用 16GB 的连续内存空间。可见，不论怎样划分，其结果都是不能接受的。因此，必须采用多级页表，将 16GB 的外层页表再进行分页，也就是将各个分页离散地分配到不相邻接的物理块中，再利用第 2 级的外层页表来映射它们之间的关系。事实上，对于 64 位机器，三级页表结构都难以适用，有些 32 位机器采用了三级页表结构，如 SUN 公司的 SPARC 处理器支持三级页表结构，Motorola 的 68030 处理器支持四级页表结构。

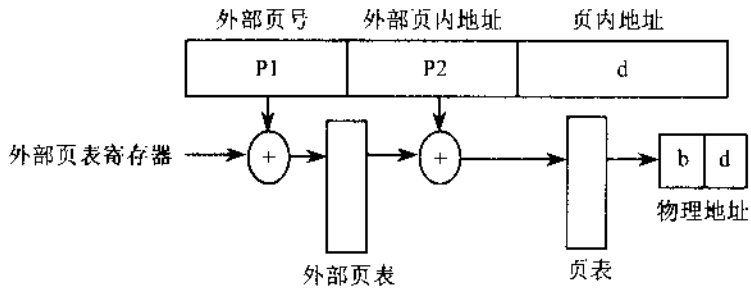


图 4-20 具有两级页表的地址变换机构

4.4 分段存储管理方式

前面介绍的存储管理方式从固定分区、动态分区到分页存储管理方式，目的主要是为了提高内存利用率，而没有考虑程序的逻辑结构。本节引入分段存储管理方式，主要是为了满足用户和编程方面的要求，其中有些要求是其他存储管理方式难以满足的。因此，分段存储管理成为当今所有存储管理方式的基础，许多高级语言编程都支持分段式存储管理。

4.4.1 分段存储管理方式的基本原理

1. 分段

在分段存储管理方式中，段是一组逻辑信息集合。例如，把作业按逻辑关系加以组织，划分成若干段，并按这些段来分配内存，这些段可以是主程序段 MAIN、子程序段、数据段和堆栈段等。

每个段都有自己的名字和长度，为了实现简单，通常用一个段号来代替段名。每个段从 0 开始编号，并采用一段连续的内存空间，各段长度不同。加载内存后，段和段之间可以是不连续的。

分段系统中的逻辑地址由段号 s 和段内偏移量 w 组成，其地址结构如图 4-21 所示。

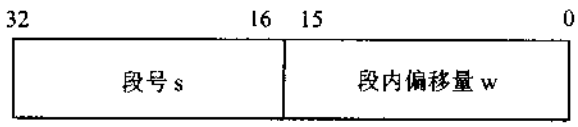


图 4-21 分段系统中的地址结构

2. 分页和分段的主要区别

分页和分段存储管理方式有很多地方相似，例如，它们在内存中的存储是离散的，都是通过地址映射机构将逻辑地址映射到物理地址中，但二者在概念上完全不同。

（1）页是信息的物理单位，分页是为了实现离散分配，提高内存利用率，便于系统管理；而段是信息的逻辑单位，每一段在逻辑上是相对完整的一组信息，如一个函数、一个过程或一个数组，分段是为了满足用户的需要。

（2）页式存储管理的逻辑空间是一维的，地址从 0 开始编号，直到末尾；而分段式存储管理作业地址空间是二维的，要识别一个地址，除给出段内地址外，还必须给出段号。

（3）物理块的长度由系统决定，是等长的；而段的长度是由具有相对完整意义的信息长度决定。

3. 基本原理

分段管理就是将作业的地址空间划分成若干个逻辑段，并且按分段进行存储分配。加载内存时，每个逻辑段占据一块连续的内存空间，但段和段之间可以是不连续的。分段的作业地址空间是二维的。与分页管理类似，可以采用动态重定位技术进行地址转换。系统为每个作业建立一张段表，每个表目至少有 4 个数据项，即段号、段长、内存始址和存取控制。其中，段号为段的序号；段长指明段的大小；内存始地址指明该段在内存中的地址；存取控制说明对该段访问的权限。

段地址转换和分页地址转换的过程基本相同，其大致过程如图 4-22 所示。

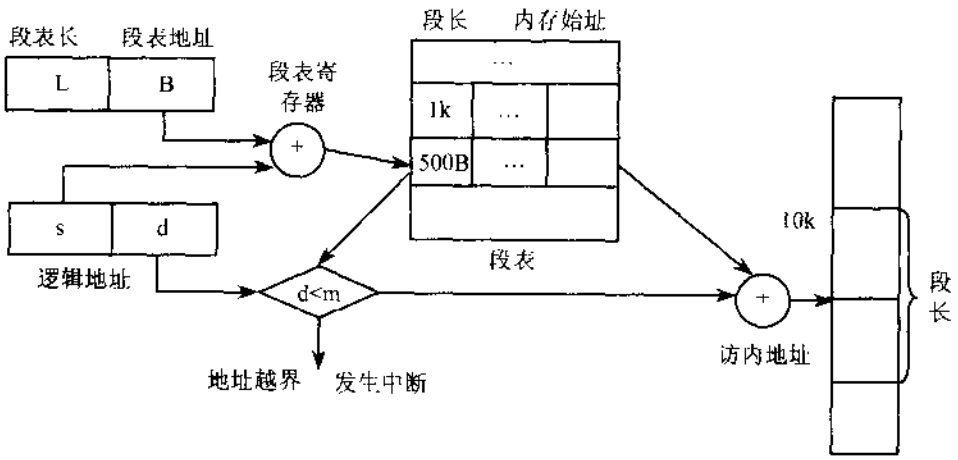


图 4-22　分段地址转换过程

（1）将逻辑地址分为两部分，段号 s 和段内偏移量 w。

（2）将该进程的段表地址寄存器中的段表内存地址 B 与段号 s 相加，得到该进程在段表中相应表项的索引值，从该表项中得到该段的长度以及该段在内存中的起始地址。

（3）将段内地址 d 与该段的内存始址相加得到访问单元的内存地址。

4.4.2 信息共享

段是按逻辑意义划分的，可以按名存取，所以，段式存储管理可以方便地实现内存的信息共享，并进行有效的内存保护。

1. 段的共享

段的共享是指两个以上的作业使用同一个子程序段或数据段，该部分在内存中只包含一个副本。具体的操作是在每个进程的段表中，用相应的表项指向共享段在内存中的起始地址，如图 4-23 所示。

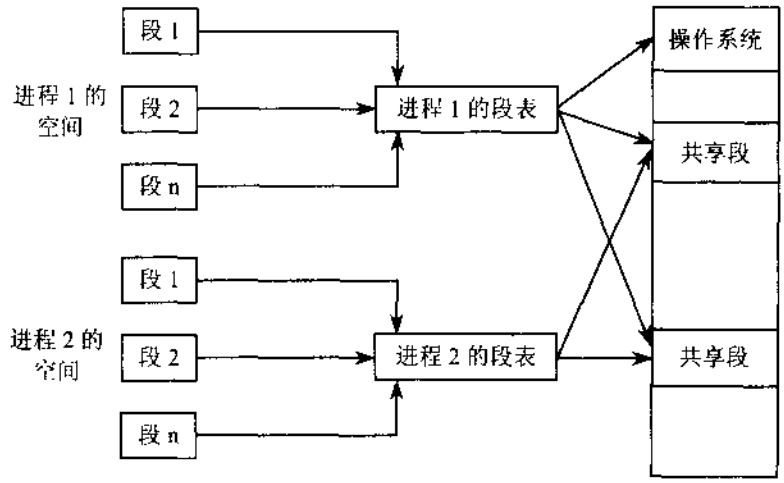


图 4-23 分段系统中段的共享

当用户进程或作业需要共享内存中某段的程序或数据时，只要用户使用相同的名字，就可以在新的段表中填入已存在段的内存起始地址，并设置一定的访问权限，从而实现段的共享。当共享此段的某进程不再需要它时，应将该段释放，取消在该进程中共享段所对应的表项。

2. 段的保护

在分段系统中，由于每个分段在逻辑上是独立的，因而比较容易实现信息保护。段的保护是实现段的共享和保证作业正常运行的一种措施。分段存储管理中的保护主要有地址越界保护和存取方式控制保护。地址越界保护是利用段表中的段长和逻辑地址中的段内相对地址相比较，如段内地址大于段长，则发出地址越界中断，系统会对段进行保护。这样，各段都限定了一定的空间，每个作业都在自己的地址空间中运行，不会发生一个用户作业破坏另一个用户作业的危险。但有的系统中允许段动态增长，在此系统中，段内相对地址大于段长是允许的，为此，段表中设置增补位，以指示该段是否允许动态增长。

4.4.3 段页式存储管理方式

分页式存储管理能有效地提高内存的利用率，分段式存储管理充分考虑程序的逻辑结构，能有效地满足用户的需要。段页式存储管理方式吸取了分页和分段存储管理两种方式

的优点，既考虑了程序的逻辑结构，又实现了不连续加载内存目的。

1. 基本思想

段页式存储管理方便使用又提高了内存利用率，是目前应用较为广泛的一种存储管理方式。它主要涉及到以下概念。

（1）采用页式存储管理的方式，将内存划分为一些大小相等的物理块。

（2）逻辑空间采用分段方式，按程序的逻辑关系把进程的地址空间分成若干逻辑段。

（3）段内分页。将每个逻辑段按页式存储管理的方式分为一些大小相等的逻辑页，页大小等于内存块大小。在每个段内，从 0 开始依次编以连续的页号。

（4）逻辑地址结构。一个逻辑地址由段号 s、段内页号 p 和页内地址 d 构成，记做 v=(s,p,d)，如图 4-24 所示。

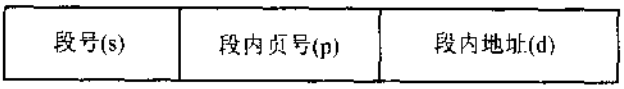


图 4-24　逻辑地址结构

（5）内存分配。内存以物理块为单位分配给每个进程。

（6）段表、页表和段表地址寄存器。为了实现从逻辑地址到物理地址的转换，系统要为每个进程或作业建立一张段表，并且还要为该作业中的每一段建立一个页表。这样，作业段表的内容是页表长度和页表地址，为了指出运行作业的段表地址，系统有一个段表地址寄存器，它指出作业的段表长度和段表起始地址。如图 4-25 给出了段表、页表和内存的关系。

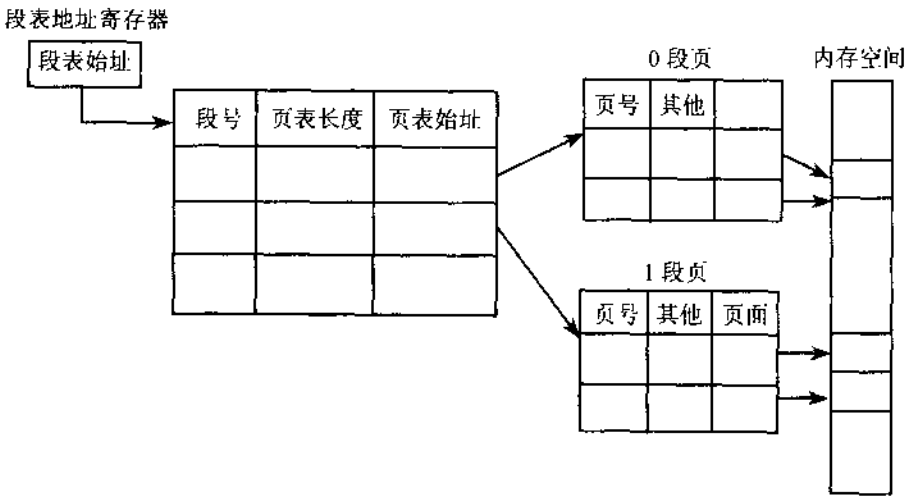


图 4-25　段页式存储管理的地址映射

在段页式存储管理系统中，面向物理实现的地址空间是页式划分的，而面向用户的地址空间是段式划分的，也就是说，用户程序被逻辑划分为若干段，每段又划分成若干页面，内存划分成对应大小的块，进程映像是以页为单位进行的，从而使逻辑上连续的段存入到分散内存块中。

2. 地址转换

在段页式系统中，一个程序首先被分成若干程序段，每一段赋予不同的分段标识符，然后，将每一段再分成若干个固定大小的页面。段页式系统中地址转换过程如图 4-26 所示。

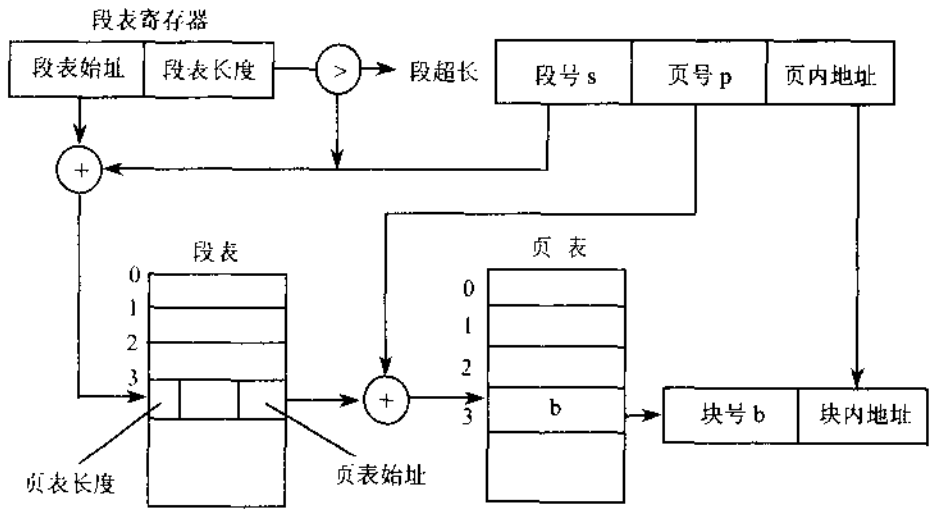


图 4-26 段页式系统中的地址转换机构

段页式存储管理方式中的地址转换过程包括如下步骤：

（1）首先利用段号 s，将它与段长 TL 进行比较，若 s<TL，表示未越界。于是地址转换硬件将段表地址寄存器的内容和逻辑地址中的段号相加，得到访问该作业段表的入口地址。

（2）将段表中的页表长度与逻辑地址中页号 p 进行比较，如果页号 p 大于页表长度，则发生中断，否则正常进行步骤（3）。

（3）按照该段的页表基地址，找到该段的页表。在该页表中，找到页号 p 所在的项，得到访问段 s 的页表和第 p 页的入口地址。

（4）从该页表对应的表项中读出该页所在的物理块号 f，再用块号 f 和页内地址 d 得到物理地址。

（5）如果对应的页不在内存，则发生缺页中断，系统进行缺页中断处理，如果该段的页表不在内存中，则发生缺段中断，然后由系统为该段在内存建立页表。

在段页式系统中，为了获得一条指令或数据，需要访问内存三次。第一次是访问段表，取得页表始地址；第二次是访问页表，获得要访问操作数的物理地址；第三次是根据物理地址取出指令或数据。为了提高访问速度，在地址变换机构中增设一高速缓冲寄存器，它的基本原理与分页和分段式的情况相似。

3. 管理算法

在地址转换过程中，软、硬件应密切配合，这在分页和分段式存储管理中已体现出来，段页式存储管理也是如此，如图 4-27 所示。

（1）链接障碍中断。这个模块的功能是实现动态链接，即给每个段一个段号，在相应的段表和现行调用表中为其设置表目，并利用段号改造链接间接字。

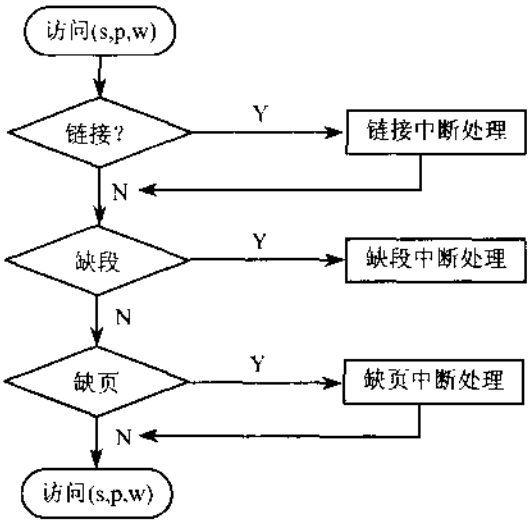


图 4-27　段页式地址转换中硬、软件的相互作用

（2）缺段中断。这个模块的功能是在系统的现行分段表中建立一个表目，并为调进的段建立一张页表，在其段表的相应表目中登记此页表的始址。

（3）缺页中断。发生缺页时进行。这个模块的功能是在内存中查找空闲的存储块，如果找到，则将该页调入内存相应空闲块；如果没找到，则调用交换算法，交换内存中的页到外存，并调进所需页面到内存，然后修改相应的页表表目。

段页式存储管理是分段技术和分页技术的结合，因而，它具备了这些技术的综合优点，便于处理变化的数据结构，段可以动态增长；便于共享和控制存取访问权限。

但是，它也有缺点，段页式存储管理增加了软件的复杂性和管理开销，也增加了硬件成本，需要更多的硬件支持；此外，各种表格要占用一定的存储空间。

4.5　虚拟存储器的基本概念

4.5.1　虚拟存储器的引入

前面讲的存储管理都要求作业一次性装入内存，但在实际运行中，作业一次性装入作业是没有必要的，因为作业装入内存后，便一直驻留在内存，占据了宝贵的内存资源，且程序的运行一般是局部性的，因此没必要在运行前将程序全部装入内存。

1. 局部性原理

早在 1968 年 P.Denning 就指出过，程序在执行时，将呈现出局部性规律，即在很短的时间内，程序的执行仅限于某个部分。相应地，它所访问的存储空间仅限于某个区域。他提出以下几个论点：

（1）程序执行时，除少部分的转移和过程调用外，大多数程序顺序执行。

（2）过程调用将使程序的执行轨迹从一部分区域转到另一部分区域。程序将在一段

时间内在这些过程内运行。

（3）程序中存在许多循环结构，它们由少数指令组成，但执行多次。

（4）程序中还包括许多对数据结构的处理。当对数组进行操作时，往往局限于很小的范围。

局限性表现在：

（1）时间的局限性。程序中某条指令一旦执行，不久又可能再被执行；某个数据结构被访问不久又可能被访问。

（2）空间的局限性。一旦某个存储单元被访问，不久后又会被访问。

2. 虚拟存储器的概念

基于局部性原理，作业在装入前，没必要一次性装入，仅将当前要运行的部分装入即可，其余的仍存放在磁盘上，程序要访问的页如果未装入内存，则将它们装入到内存，继续执行。若内存已满，则将内存中暂时不用的页调出内存。释放所占内存空间，将要访问的页在内存中运行，这样就可以将一个较小的程序调到一个较大的内存空间中运行。从用户角度看，该系统所具有的内存容量比实际的内存容量要大得多，人们把这样的存储器称为虚拟存储器。

虚拟存储器是作业运行时，仅把部分运行到的程序和数据装入内存便可以运行的虚拟存储系统。具体地讲，虚拟存储器是指具有请求调入功能和置换功能，并能从逻辑上对内存空间进行扩充的一种存储器系统。虚拟存储器是一种逻辑上的存储器，它使用户感觉到它占有比实际内存大得多的内存空间。

必须指出，实现虚拟存储技术需要一定物质基础，第一是需要相当容量的辅存，以便足以存放多用户作业的地址空间；第二是要一定容量的主存；第三是需要地址变换机构。所以设计虚拟存储器时，应在可能的情况下力求地址变换机构快速地进行。

4.5.2 虚拟存储器的实现方法及其特征

虚拟存储器的实现方法包括请求分页系统和请求分段系统两大类。

1. 请求分页系统

请求分页系统是在分页存储管理方式的基础上增加了请求调页功能、页面置换功能所形成的页式虚拟存储系统。程序开始只是装入若干页的用户程序和数据，便可以启动运行，在以后的运行过程中，访问到其他逻辑页时，再陆续将所需的页调入内存。请求调页和置换时，需要以下软、硬件的支持：

（1）请求分页的页表机构。

（2）缺页中断机构。

（3）地址变换机构。

2. 请求分段系统

请求分段系统是在分段存储管理方式的基础上增加了请求调段及分段置换功能而形成的段式虚拟存储系统，它允许只装入部分程序和数据，运行时动态调入。实现请求分段，同样需要以下机构的支持：

（1）请求分段的段表机制。

（2）缺段中断机构。

（3）地址变换机构。

3. 虚拟存储器的特征

虚拟存储器具有虚拟性、离散性、多次性及对换性等特征，其中最重要的特征是虚拟性。

（1）虚拟性。虚拟性是指能够从逻辑上扩充内存容量，使用户所看到的内存容量远大于实际的内存容量，这是虚拟存储器所表现出的最重要的特征，也是虚拟存储器最重要的目标。

（2）离散性。离散性是指内存分配时采用离散分配的方式，没有离散性就不可能实现虚拟存储器。如果采用连续分配方式，需将作业装入到连续的内存区域，这样需要连续地一次性申请一部分内存空间，以便将整个作业先后多次装入内存。如果仍然采用连续装入的方式，无法实现虚拟存储功能，只有采用离散分配方式，才能为它申请内存空间，以避免浪费内存空间。

（3）多次性。多次性是指一个作业被分成多次调入内存运行。作业在运行时，只将当前运行的那部分程序和数据装入内存，以后再需要哪部分时，再从外存调入内存。

（4）对换性。对换性是指允许在作业运行过程中换进换出。允许将暂时不用的程序和数据从内存调至外存的对换区，以后需要时再从外存调到内存。

4.6　请求分页

4.6.1　请求分页的实现

请求分页式存储管理是建立在页式存储管理的基础上，结合虚拟存储系统原理实现的，是目前常用的一种实现虚拟存储器的方式，它既需要硬件支持又需要软件支持。换进、换出的基本单位是固定长度的页面。

1. 页表机制

分页系统中地址映像是通过页表实现的。在请求分页系统中，页表项包括下列信息：

逻辑页号	物理块号	状态位 P	访问字段 A	修改位 M	外存地址

（1）状态位 P，用于指示该页是否已调入内存。

（2）访问字段 A，用于记录本页在一段时间内被访问的次数，或最近多长时间没被访问，供置换算法换出页面时参考。

（3）修改位 M，表示该页在调入内存后是否被修改过。由于内存的每一页在外存都有备份，所以在该页被唤出时，若查出该页被修改过，则将该页再写回外存；若调入内存后未被修改过，则直接进行覆盖，无需写回。

（4）外存地址，用于指出该页在外存的地址，如果外存采用磁盘，则指磁盘块号。

2. 缺页中断机构

在请求分页系统中，若所访问的页不在主存，便产生一缺页中断。缺页中断和一般中

断相比具有明显的区别，主要表现在以下几个方面：

（1）在指令执行期间产生和处理中断信号。通常，CPU 都是在一条指令执行完后，再去判断是否有中断。若有则响应中断，没有则执行下一条指令。

（2）一条指令在执行期间可能产生多次中断。系统中的硬件应能保存多次中断时的状态，并保证最后能返回到中断前的缺页中断的指令处，继续执行。如图 4-28 所示是缺页中断处理的流程。

3. 地址变换过程

请求分页系统中的地址变换机构是在分页系统的地址变换基础上，为实现虚拟存储器而增加某些功能实现的。当出现缺页中断时，保留当前 CPU 的状态，在外存找到所需页面，按照一定的页面转换算法，将新的页面调入内存，然后恢复 CPU 的状态，继续执行。

请求分页的地址变换过程如图 4-29 所示。

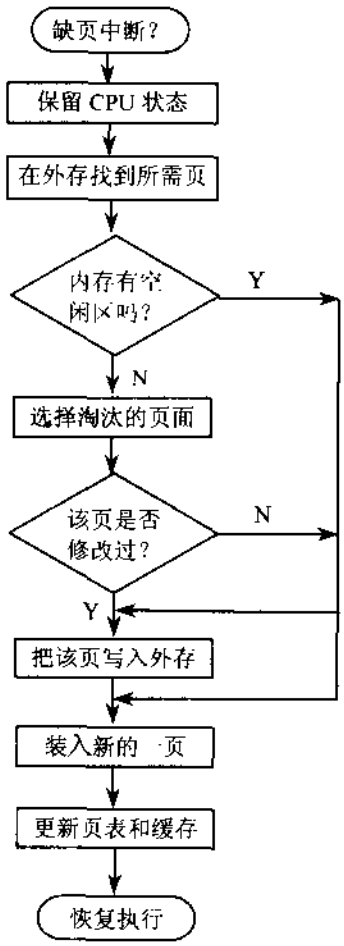


图 4-28 缺页中断处理过程

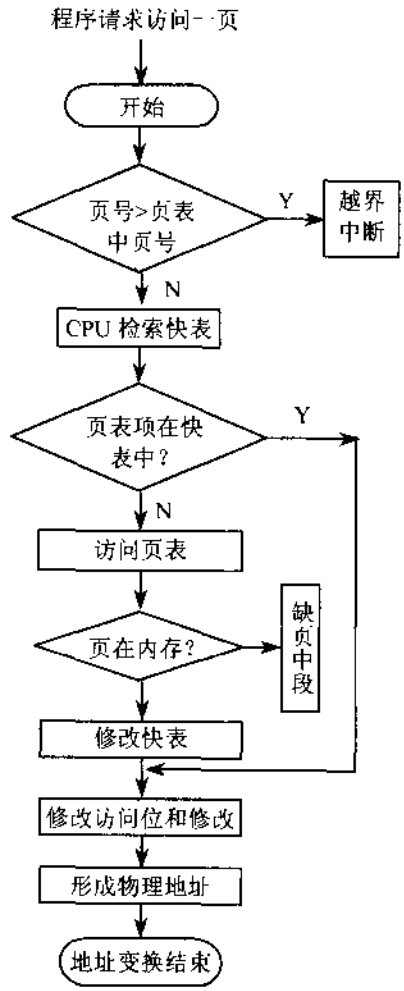


图 4-29 请求分页系统的地址变换过程

4.6.2　页面置换算法

进程运行过程中，若其所要访问的页面不在内存，则需要将它调入内存。当内存没有空闲空间时，必须从内存空间中调出一页。调出页面的原则必须根据一定的算法进行，该算法称为页面置换算法。置换算法的好坏直接影响到系统的性能。不合适的算法可能会使系统产生“抖动”，即刚换出的页很快又被访问，需重新调入；选出一页调出，而刚调出的页，可能很快又要被访问，又需要将它调入。如此频繁地更换页面，以致一个进程在运行中，大部分时间花费在页面置换上。

下面介绍几种常用的页面置换算法。

1. 先进先出页面置换算法（FIFO）

采用该算法，当需要置换一个页面时，总是置换最先进入内存的那个页面。先进先出算法是一种最简单的置换算法，实现时只需将调入内存的页面按先后顺序排成一个队列。当需要置换一个页面时，总是将年龄最老的那个页面淘汰出内存。

假设某进程的最大页面数为 3，对于如图 4-30 所示的页面走向，其页面失效的次数为 15，页面失效率为 3/4。FIFO 置换算法是易于理解和实行的。实行时，只要建立一个 FIFO 队列，并规定最新进入的页面总排在队列末尾，而需要置换时，总是把当前队首的那个页面换掉。

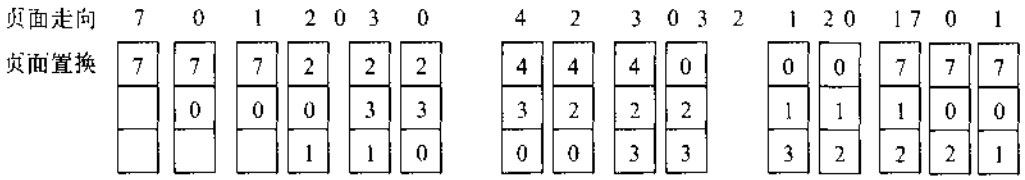


图 4-30　页面置换算法

然而，FIFO 算法有可能产生异常现象，在相同的页面走向下当分给某一进程的页面数增多时，页面失效不但不降，反而更加严重。

2. 最近最久未使用页面置换算法（LRU）

该算法出现缺页中断时，总是选择最近一段时间内，最长时间没有被访问过的页面，将它换出外存。LRU 算法是一种比较好的页面置换算法，能适应各种类型的程序，但实现起来有相当大的难度，因为它要求系统具有较多的支持硬件。所需要解决的问题是一个进程在内存中的各个页面有多长时间未被进程使用；如何快速知道哪一页是最近最久未使用的页面。为此，需要利用以下两类支持硬件。

（1）寄存器。用于记录某进程在内存中各页的使用情况。所以，需要为每个内存中的页面配置一个移位寄存器，可表示为 $R=R_{n-1}R_{n-2}R_{n-3}\ldots R_2R_1R_0$。

当进程访问某物理块时，要将相应寄存器的 R_{n-1} 位置成 1。此时，定时信号将每隔一定时间将寄存器右移一位。如果把 n 位寄存器的数看作是一个页符号的整数，那么最小数值的寄存器所对应的页面就是最近最久未使用的页面，如图 4-31 所示。

R 实页	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0
1	0	1	0	1	0	0	1	0
2	1	0	1	0	1	1	0	0
3	0	0	0	0	0	1	0	0
4	0	1	1	0	1	0	1	1
5	1	1	0	1	0	1	1	0
6	0	0	1	0	1	0	1	1
7	0	0	0	0	0	1	1	1
8	0	1	1	0	1	1	0	1

图 4-31 某进程具有 8 个页面时的 LRU 访问情况

（2）栈。利用一个特殊的栈来保存当前使用的各个页面的页号。每当进程访问某页面时，便将该页面的页面号从栈中移出，将它压入栈顶。因此，栈顶始终是最新被访问页面的编号，而栈底则是最近最久未使用页面的页面号。假设有一进程所访问页面的页号序列为 4，7，0，7，1，0，1，2，1，2，6。随着进程的访问，栈中页面号的变化情况如图 4-32 所示，当访问页面 6 时，发生缺页，此时页面 4 是最近最久未被访问的页，应将它置换出内存。

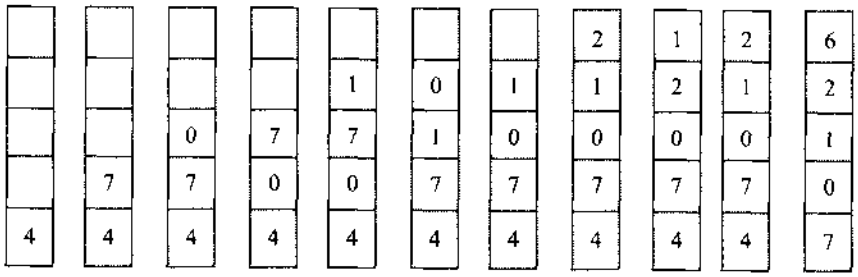


图 4-32 用栈保存当前使用页面时栈的变化情况

3. 最佳置换算法（OPT）

最佳置换算法是由 Belady 于 1966 年提出的一种理论上的理想算法，它所选择的被淘汰的页面将是最长时间不被使用的。对于固定分配页面方式，采用最佳置换算法可以保证最低的缺页率。但由于人们目前还无法预知一个进程在内存的若干个页面中，哪个页面是未来最长时间内不再被访问的，因而该算法基本上是很难实现的，但可以利用该算法评价其他算法。

例 3：已知系统为一个作业分配了三个物理块，该作业的页面走向为：

7，0，1，2，0，3，0，4，2，3，0，3，2，1，2，0，1，7，0，1

计算该作业的页面淘汰次序及缺页次数。

解：进程运行时先将 7，0，1 三个页面装入内存。当进程访问页面 2 时，将会产生缺页中断，此时，操作系统根据最佳置换算法，将选择页面 7 淘汰，这是因为页面 0 将作为

第 5 次被访问的页面，页面 1 是第 14 次被访问的页面，页面 7 将是第 18 次要被访问的页面。很显然要淘汰最久才访问的页面。

页面置换过程如图 4-33 所示。

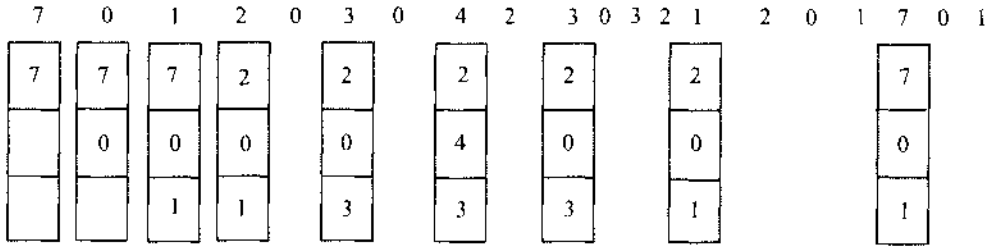


图 4-33　利用最佳页面置换算法时的置换图

4.7　请求分段存储管理

请求分页内存管理方式是以页面为单位进行换入、换出的，而在段式存储管理基础上实现的虚拟存储器，则以段为单位进行换入、换出。在请求分段系统中，程序运行之前先将开始运行时用到的段调入内存，便可以运行。当访问到的段不在内存时，再将该段调入内存。

4.7.1　请求分段的实现

为了实现请求分段存储管理方式，应在系统中配置多种硬件机构，包括段表机制、缺段中断机构及地址变换机构。

1. 段表机制

请求分段式管理用的主要数据结构是段表。段表结构如下：

段名	段长	段的基址	存取方式	访问字段 A	修改字段 M	存在位 P	增补位	外存起址

在段表项中，除了段名、段长、段在内存的起始地址外，还增加了以下几项：

（1）存取方式，用于标识本分段的存取属性是执行、只读还是可读可写。

（2）访问字段 A，其含义与请求分段的相应字段相同，用于记录被访问段的频率。

（3）修改位 M，表示该页进入内存后是否被修改过。

（4）存在位 P，指示本段是否已调入内存。

（5）增补位，表示该段在运行过程中是否动态增长。

（6）外存始址，指示本段在外存中的起始地址，即起始磁盘块号。

2. 缺段中断机构

请求分段系统采用的是请求调段策略。每当进程要访问的段不在内存时，就产生缺段中断。缺段中断的处理过程如下：

（1）判断虚段 S 是否在内存。

（2）若不在，则让请求进程处于阻塞状态。

（3）判断内存中是否有空闲区域。

（4）若有空闲区域，则从外存读入段 S，修改段表和空闲区域表，唤醒请求进程。

（5）若没有空闲区域，判断总的空闲区是否满足，若满足，则将各空闲区合并，调入段 S，修改段表。若总的空闲区不够，则淘汰一个或几个段，形成一个合适的空闲区，再调入段 S。缺段中断流程图如图 4-34 所示。

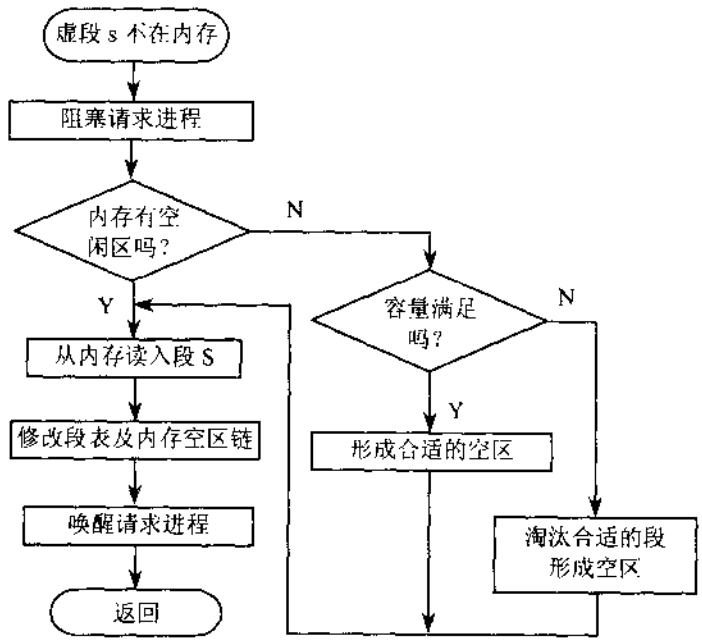


图 4-34 缺段中断流程图

3. 地址变换机构

请求分段系统的地址变换机构是在分段系统地址变换的基础上形成的。当所要访问的段不在内存时，将所缺的段调入内存，并修改段表，再利用段表进行地址变换。变换过程如下：

（1）有一个要访问的逻辑地址格式为[S|w]，其中 S 为段号，w 为段内偏移量。

（2）判定若 w 的值大于段长，则进行越界中断处理。

（3）若 w 的值小于段长，则检查是否为合法的存取方式，若不是，则进行分段保护中断处理。

（4）若其他两种情况合法，则判断段 S 是否在主存。若在，则修改访问位，将形成主存的地址段的始地址加上偏移量。

地址变换机构流程图如图 4-35 所示。

例 4：现有一个作业，在段式存储器管理的系统中已为其分配内存，建立的段表内容如表 4-2 所示。

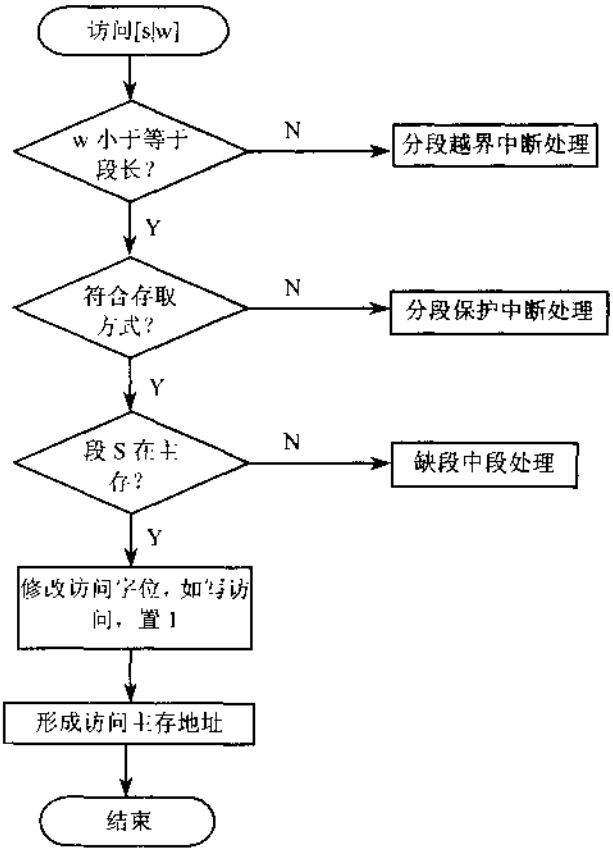


图 4-35　请求分段系统的地址变换机构流程图

表 4-2　例 4

段号	主存起始地址	段长度
0	1200B	40 B
1	7600B	30 B
2	4800B	20 B
3	3700B	20 B

已知各操作数的逻辑地址分别为(2,15)、(0,60)、(3,18)，其中，括号中第一个元素为段号，第二个元素为段内地址。

试求各操作数的物理地址。

解：段式存储器管理的地址转换过程如下：

（1）根据逻辑地址中段号查段表的相应栏目。

（2）根据段内地址<段长度，检查地址是否越界。

（3）若不越界，则物理地址=该段的主存起始地址+段内地址。

逻辑地址(2,15)查段表得到段长度为 20B，段内地址 15<20，地址不越界；段号 2 查段

表得到段首地址为 4800B，于是该操作数在内存的物理地址为 4800+15=4815B。

逻辑地址(0,60)查段表得到第 0 段的段长为 40B，段内地址 60>40，地址越界，系统发出“地址越界”中断。

逻辑地址(3,18)查段表得到段长度为 20B，段内地址 18<20，地址不越界；段号 3 查表得段首地址为 3700B，于是绝对地址=3700+18=3718B。

4.7.2 段的共享与保护

请求分段存储管理方式便于实现分段的共享和保护。段的共享和保护通常通过段表来实现。

1. 段表

为了实现分段共享，可在系统中配置一张段表，所有的共享段在段表中占一项。段表项的结构如图 4-36 所示。

段名	段长	内存始址	状态	外存始址
共享进程计数 COUNT				
状态	进程名	进程号	段号	存储控制
…				

图 4-36 共享段表

（1）共享进程计数 count。为了记录有多少个进程需要共享该段，特别设置了一个整型变量 count，用以记录系统中共享该段的进程数。当进程不再需要该段时，可释放该段，并收回该段所占用的内存空间，count 减 1。

（2）存取控制字段。对于一个共享段，应给不同的进程以不同的存取权限。例如，对于文件主，通常允许读和写；而对其他进程，只允许读或执行。

（3）段号。对于同一个共享段，不同的进程可以使用不同的段号去共享该段。

2. 共享段的分配与回收

（1）共享段的分配。由于共享段是供多个进程共享，因此，共享段的内存分配方式和非共享段的内存分配方式是不同的。在分配共享段时，对第一个使用该段的进程，系统将该共享段调入内存，为共享段加一表项，并填写有关表项，把 count 加 1。当其他的进程要共享该段时，不再需要调入内存，只需加一表项，填入该共享段的物理地址；在共享段的段表中，填上进程名、存取控制权限，再执行 count=count+1 操作，表明增加一个进程共享该段。

（2）共享段的回收。当共享某段的某进程不再需要该段时，则将该段释放，包括取消在该进程段表中共享该段所对应的表项，以及执行 count:=count-1 操作。若减 1 后结果为 0，则由该系统收回该共享段的物理地址。

3. 分段保护

在分段系统中，由于每个分段逻辑上是独立的，并且加载内存后可以分别在不同的区

域，因此比较容易实现信息保护。目前，采用以下几种方式对段进行保护。

（1）越界检查。在段表寄存器中，存放着段表的长度信息；同样，在段表中也为每个段表设置段长度字段。进行访问时，将操作数逻辑地址空间的段号和段表长度进行比较，若等于或大于段表长度，则发出越界中断信号；其次，还要检查段内地址是否等于或大于段长。若大于段长，将产生越界中断信号，从而保证各进程只能访问自己的地址空间。

（2）存取控制检查。在段表的每个表项中，设置一个存取控制字段，通常的访问方式有：

1）只读。只允许进程对该段中的程序和数据进行访问。

2）只执行。只允许执行，不允许读写。

3）读/写。允许进程对该段进行读写控制。不同的访问对象，赋予不同的权限。

（3）环保护机构。它是一种较完善的保护机构。该机制规定低编号的环具有较高的优先权。操作系统居于 0 环内；某些重要的系统软件占据中间环，而一般的应用程序位于外环。

程序访问和调用遵循以下规则：

1）一个程序可以访问驻留在相同环或较低环中的数据。

2）一个程序可以调用驻留在相同环或高特权环中的服务。

4.8　Linux 系统的内存管理方法

Linux 采用“按需调页”算法，支持三层管理策略。由于 Intel CPU 在硬件级提供了段式存储管理和二层页式存储管理功能，Linux 操作系统作为一种软件，必须与之兼容。Linux 根据 Intel CPU 的要求，最低限度地设置与段相关的结构和初始化程序，但实质上是放弃了段式存储管理。Intel 微机上的 Linux 系统考虑到 CPU 的限制，将第二层上的页式管理（pmd）与第一层上的页式管理（pgd）合并，因此真正发挥作用的是以页目录和页表为中心的数据结构和函数。

4.8.1　Linux 的分页管理机制

在 Linux 中，每个用户进程都可以访问 4GB 的线性虚拟内存空间。其中，0～3GB 的虚拟地址空间是用户空间，用户进程可以直接对其进行访问；3GB～4GB 的虚拟内存地址空间为内核态空间，存放仅供内核态访问的代码和数据，用户态进程不可访问。当用户进程通过中断或系统调用访问内核态空间时，就会触发处理特权级转换，即从操作系统的用户态转换到内核态。

所有进程从 3GB 到 4GB 的虚拟空间是一样的，有相同的页目录项和页表，对应同样的物理内存段。Linux 以此方式让内核态进程共享代码段和数据段。

内核态虚拟空间从 3GB 到 3GB+4MB 的一段被映射到物理空间 0～4MB。因此，进程处于内核态时，只要通过虚拟空间 3GB～3GB+4MB 段，即访问了物理空间 0～4MB 段。

上述两种空间对用户进程来说都是透明的。用户进程所访问的内存地址是连续的 4GB 线性虚拟地址，因此，首先关心的是 Linux 如何划分虚拟空间。

Linux 采用“按需调页”技术管理虚拟内存。标准 Linux 的虚存页表应为三级页表，依次为页目录（PGD，Page Directory）、中间页目录（PMD，Page Middle Directory）和页表（PTE，PageTable），如图 4-37 所示。

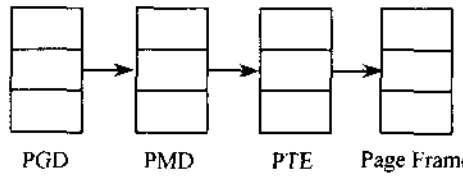


图 4-37 inux 的三级页表结构

在 Intel 微机上，Linux 的页表结构实际为两级。80386 体系结构的页管理机制中的页目录是 PGD，页表是 PTE，而 PGD 和 PMD 实际上合二为一。在用户进程中用到的与内存管理有关的数据结构是 mm_struct 结构，此结构中包含了用户进程中与存储有关的信息。mm_struct 结构定义如下：

```
struct mm_struct{
int count;
pgd_t *pgd;
unsigned long context;
unsigned long start_code,end_code,start_data,end_data;
unsigned long start_brk,brk,start_stack,start_mmap;
unsigned long arg_start,arg_end,env_start,,env_end;
unsigned long rss,total_vm,locked_vm;
unsigned long def_flags;
struct vm_area_struct *mmap;
struct vm_area_struct *mmap_avl;
struct semaphore mmap_sem;
}
```

pgd：进程页目录的起始地址，如图 4-38 所示。

start_code、end_code：进程数据段的起始地址和结束地址。

env_start、env_end：进程环境区的起始地址和结束地址。

rss：进程内容驻留在物理内存的页面总数。

mmap：指向 vma 段双向链表的指针。

mmap_avl：指向 vma 段 AVL 树的指针。

mmap_sem：对 mmap 操作的互斥信号量，由 down()和 up()更改。

此外，每一个进程都有一个页目录，存储该进程所使用的内存页面情况。Linux 按照“按需调页”的原则，只分配必需的内存页面，从而避免了页表过多占用存储空间的情况出现。例如，系统调用 fork 分配内存页面的情况如下：

（1）进程控制块 1 页。

（2）内核态堆栈 1 页。

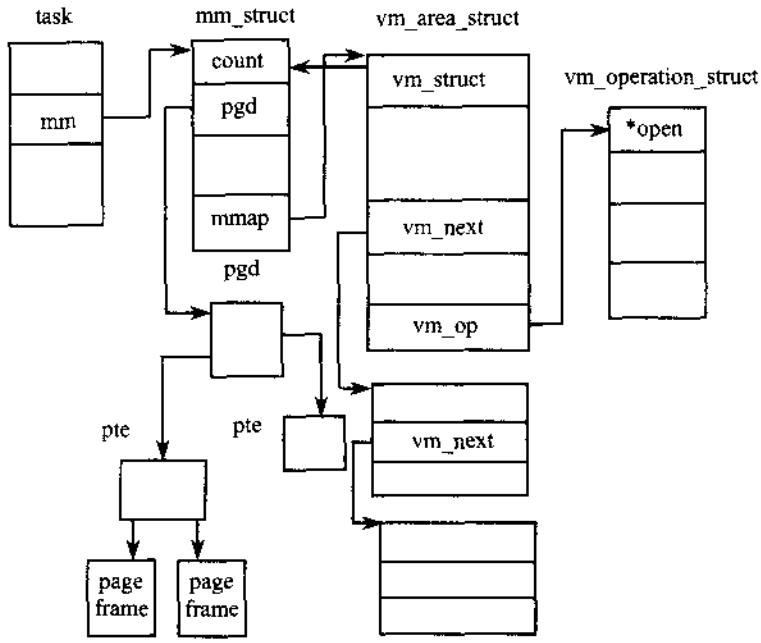


图 4-38　进程的虚存管理数据结构

（3）页目录 1 页。

（4）页表占几页。

系统调用 exec 分配内存页面的情况如下：

（1）可执行文件的文件头 1 页。

（2）用户堆栈 1 页或几页。

这样，当进程开始运行时，如果执行代码不在内存中，将产生第一次缺页中断，让操作系统分配内存页面，并将执行代码装入内存中。此后，按需要逐渐分配更多的内存页面，并参与页面调度。当系统内存不足时，由操作系统决定是否将该进程的一部分页面换出到磁盘交换区或交换文件中。进程终止时，操作系统释放所有该进程占用的资源，包括内存页面。

4.8.2　虚存段的组织与管理

用户共有 4GB 的虚存空间，但并不是 4GB 空间都可以让用户进程读写或申请使用。用户进程实际可申请的虚存空间为 0～3GB。在用户进程创建时，已由系统调用 fork()的执行函数 do_fork()将内核的代码段和数据段映射到 3GB 以后的虚存空间，供内核态进程访问。所有进程的 3GB～4GB 的虚存空间的映像都是相同的，并以此方式共享代码段和数据段。

为了能以自然的方式管理进程虚存空间，Linux 定义了虚存段（vma，virtual memory

area），一个 vma 段是某个进程的一段连续的虚拟空间，在这段虚存空间的所有单元拥有相同的特征。例如，属于同一个进程，具有相同的访问权限，同时被锁定，同时受保护等。

（1）虚存段的数据结构。

```
vm_area_struct{
struct mm_struct *vm_mm;
unsigned long vm_start;
unsigned long vm_end;
pgprot_t vm_psge_prot;
unsigned short vm flags;
short vm_avt_height;
struct vm_area_struct *vm_avl_left;
struct vm_area_struct *vm_avl_right;
struct vm_area_struct *vm_next;
struct vm_area_struct *vm_next_share;
struct vm_area_struct *vm_next_share;
struct vm_operation_struct *vm_ops;
unsigned vm operation struct *vm_ops;
unsigned long vm_offset;
struct inode *vm_inode;
unsigned long vm_pte;
};
unsigned long vm_start,vm_end;
```

vma 描述的虚拟内存段始于 vm_start，终于 vm_end。

（2）struct inode *vm_inode;

如果 vma 段的内容是关于磁盘文件或设备文件的，vm_inode 指向该文件的 inode 结构，否则，vm_inode 为 NULL。

（3）unsigned long vm_offset;

如果 vma 段的内容是关于文件的，则 vm_offset 就是该段内容相对于文件起始位置的偏移量。如果 vma 段的内容是关于共享内存的，则 vm_offset 是 vma 段的起始地址 vm_start 相对于共享段始地址的偏移量。

（4）struct vm_operation_struct *vm_ops;

如果将 vma 段理解成“对象”的话，vm_ops 规定了该对象的操作方法。这些操作包括 open、close、swapout 等。（见 include/Linux/mm.h。）

进程通常占用几个 vma 段，分别用于代码段、数据段、堆栈段等。属于同一个进程的 vma 段通过 vm_next 指针链接，组成链表，如图 4-39 所示。Struct mm_struct 结构的成员 struct vm_area_struct *mmap 表示进程的 vma 链表的表头。为了提高对 vma 段的查询、插入、删除操作的速度，Linux 同时维护了一个 AVL 树。在树中，所有 vm_area_struct 虚存段均由左指针 vm_avl_left 指向相邻的低地址虚存段，右指针 vm_avl_right 指向相邻的高地址虚存段，如图 4-39 所示。Struct mm_struct 结构的成员 struct vm_area_struct *mmap_avl 表示进程的 AVL 树，vm_avl_height *mmap_avl 表示进程 AVL 树的根，vm_avl_height 表示 AVL 树的高度。

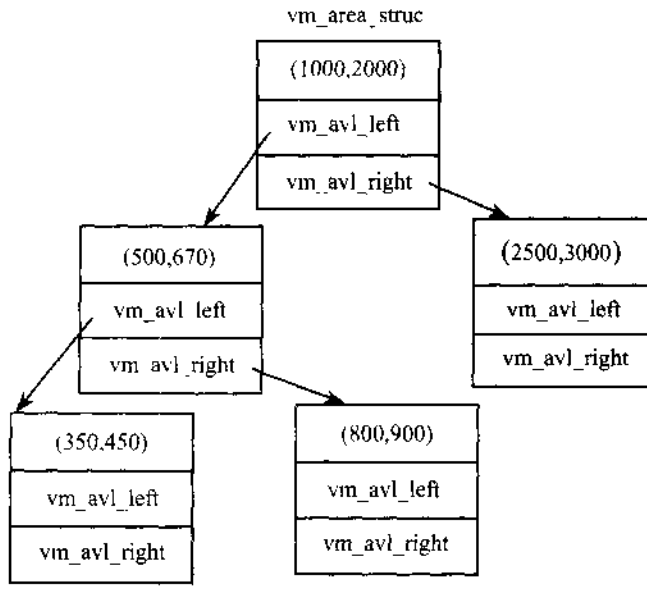


图 4-39　进程的 AVL 树

4.8.3　内存的共享和保护

Linux 中内存共享以页表的形式实现，共享该页的各进程的页表表项直接指向共享页，如图 4-40 所示。这种结构不需要设立共享页表，节约内存的占用，但效率较低。当共享页状态发生变化时，共享该页的各进程的页表均需修改，并要多次访问页表。

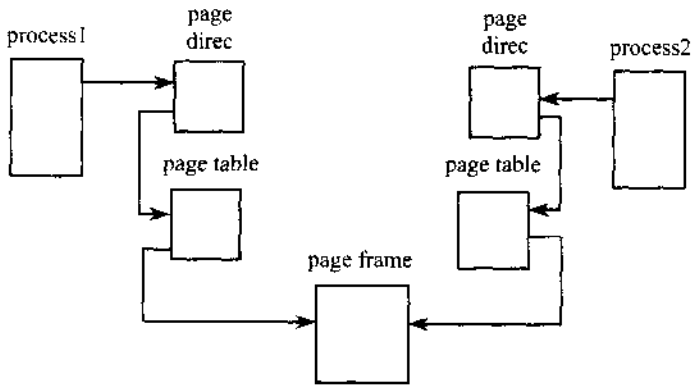


图 4-40　Linux 页共享结构

Linux 可对虚存段中任意部分加锁或保护。对进程的虚拟地址加锁，其实质就是对 vma 段的 vm_flags 属性与 VM_LOCKED（0X2000）进行“或”操作。虚存加锁后，它对应的物理页面驻留内存，不再被页面置换程序换出。加锁操作共有 3 种：

Mlock：对指定的一段虚拟空间加锁。

Munlock：对指定的一段虚拟空间解锁。

Mlockall：对进程的所有虚拟空间解锁。

对进程的虚拟地址施行地址保护措施就是重新设置 vma 段的访问权限，其实质是对 vma 段的 vm_flags 属性重置 PORT_READ(0x1)、PORT_WRITE(0X2)和 PORT_EXEC(0x4)，并重新设定 vm_page_port 属性。与此同时，对虚拟地址范围内的所有页表项，其访问权限也跟着调整。保护操作由系统调用 mprotect 实施。

虚存段加锁、保护操作可以有 4 种方式：

- 对整个虚存段加锁或保护。
- 对虚存段前部加锁或保护。
- 对虚存段后部加锁或保护。
- 对虚存段中部加锁或保护。

4.8.4 内存空间管理

尽管 Linux 采用虚拟存储管理策略，有些申请仍然需要直接分配物理空间。例如，为刚创建的进程分配页目录，为装入进程的代码段分配空间，为 I/O 操作准备缓冲区等。物理内存以页帧为单位，页帧的长度固定，等于页长，对于 Intel CPU 默认为 4KB。

Linux 对物理内存的管理通过 mem_map 表描述。mem_map 在系统初始化时，由 free_area_init()函数创建，见图 4-41 所示。

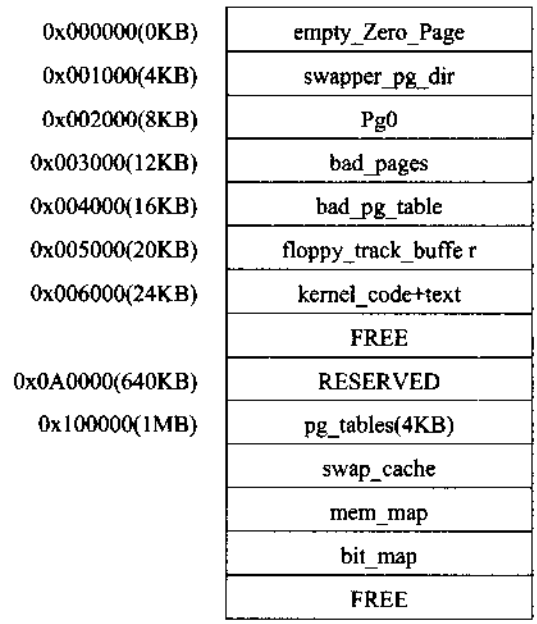


图 4-41 mem_map 表在物理内存的位置

4.8.5 空闲内存管理

在物理内存低端，紧跟 mem_map 表的 bitmap 表以位示图方式记录了所有物理内存的

空闲情况。与 men_map 一样，bitmap 表在系统初始化时由 free_area_init()函数创建，如图 4-42 所示。

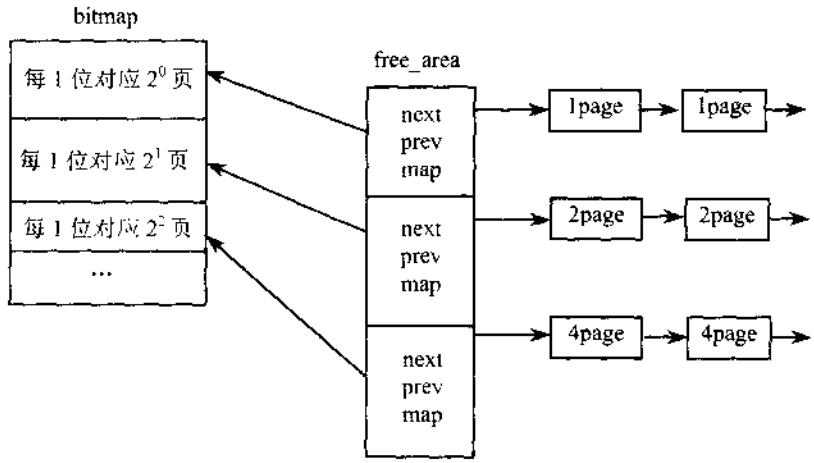


图 4-42　bitmap 表及其与 free area 的关系

与一般性位图不同的是，bitmap 表分割成 NR_MEMLISTS（默认值为 6）。首先是第 0 组，初始化时设定了长度为（end_mem-start_men）/PAGE_SIZE/2^{0+3}，每位表示 2^0 个页帧的空闲情况，置位表示已被占用。接着是第 1 组，初始化时设定了长度为（end_mem-start_mem）/PAGE_SIZE/2^{1+3}，每位表示连续 2^1 个页帧的空闲情况，置位表示其中 1 页或 2 页已被占用。类似地，对第 I 组，初始化时设定了长度为（end_mem-start_mem）/PAGE_SIZE/2^{i+3}，每位表示连续的 2^i 个页帧的空闲情况，置位表示 1 页或几页已被占用。例如，对第 5 组，某位对应的连续的 32 页帧中只要有一位被占用，此位即置 1，只有当所有的 32 页帧全部被收回后才清零。Linux 的内存空闲情况由 free_area_struct 结构类型的数组元素构成，每个元素作为空闲块链表的表头。

```
Struct free_area_struct{
struct page *next; /*此结构的 next、prev 指针与 struct page 匹配*/
struct page *prev;
unsigned int *map; /*指向 bitmap*/
};
static struct free_area_struct  free_area[NR_MEM_LISTS]
```

4.8.6　内核态内存的申请与释放

内核态内存是用来存放 Linux 内核系统数据结构的内存区域，处于进程虚拟空间的 3GB～4GB（准确地说是 3GB+high_memory，其中 high_memory 是系统在启动阶段测得物理内存的实际容量）范围内。

以 Intel 微机为例，blocksize 表的表项有 32、64、128、252 等内容。

```
Static const unsigned int blocksize[]={
32,64,128,252,508,1020,2040,4096-16,8192-16,16384-16,32768-16,65536
```

```
-16,131072-16,0
        /*表示 blocksize 表的结束*/
        };
```

在 blocksize 中能够申请到的块的大小近似于 2 的次幂，所差的恰好是管理数据结构所占的字节数，如 16 是 struct page_descriptor 结构的长度。

Size 表管理那些从物理空间 free_area 申请到的用于内核分配的空闲块。Size 表的元素类型是 struct size_descriptor，每个表项实质上就是两个由等长空闲块组成的链表的表头，一个是 DMA 可访问的，另一个是一般性的。

```
Struct size_descriptor{
{struct page_descriptor *firstfree;/*空闲块链表首址*/
struct page_descriptor *dmafree;/*DMA 可访问的空闲块链表首址*/
int nblocks;   /*本页包含此长度空闲块的数目*/
int nmallocs;/*本页已分配块数*/
int nfrees/*本页空闲块数*/
int nbytesmalloced;/*本页已分配字节数*/
int npages;/*本链表的空闲页数*/
unsigned long gfporder;/*2**gfporder 等于空闲块所需的页数*/
```

管理内核空间空闲块的数据结构及相互关系如图 4-43 所示。

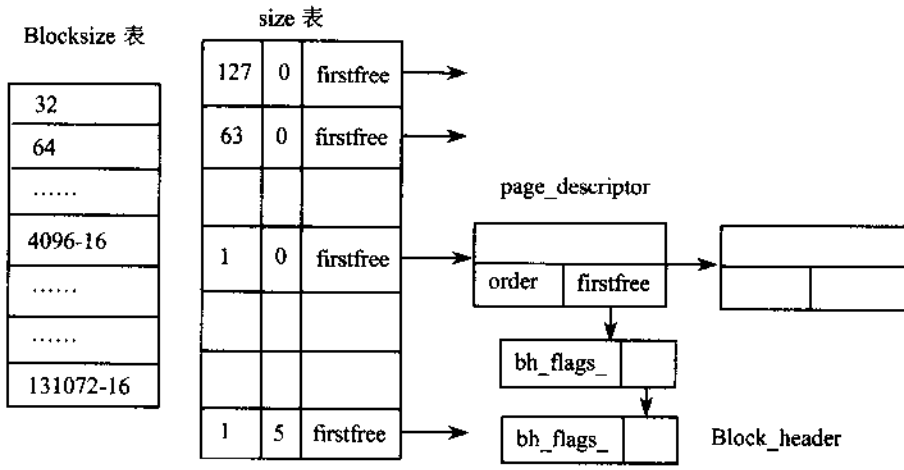


图 4-43 管理内核空间空闲块的数据结构及相互关系

4.8.7 用户态内存的申请与释放

Linux 用 kmalloc()函数和 kfree()函数提供内核内存申请、释放的接口，它还实现另一种虚拟空间的申请、释放界面，就是 vmalloc()和 vfree()。

由 vmalloc()分配的存储空间在进程的虚拟空间是连续的，但它对应的物理内存仍需经缺页中断后，由缺页中断服务程序分配，所分配的物理页帧不是连续的。这些特征和访问用户内存相似，所以不妨把 vmalloc()和 vfree()称做用户态内存的申请和释放界面。

可分配的虚拟空间在 3GB+high_memory+HOLE_8MB 以上的高端，由 vmlist 链表管理。

3GB 是内核态赖以访问物理内存的始址。high_memory 是安装在计算机中实际可用的物理内存的最高地址。因此 3GB+high_memory 也是（从虚拟空间中看到的）物理内存的上界。HOLE_8MB 则是长度为 8MB 的“隔离带”，起越界保护作用。这样，vmlist 管辖的虚拟空间既不与进程用户态 0～3GB 的虚拟空间冲突，也不与进程内核态映射到的 3GB～3GB+high_memory 的虚拟空间冲突。

尽管 vmalloc 返回高于任何物理地址的高端地址，但因为 vmalloc 同时更改了页表甚至页目录，处理器仍能正确访问这些高端连续地址。（页目录和页表把高端的虚拟地址映射到实际的物理地址）。内核态程序与用户态程序共享同一个页目录和一组页表，因而内核态程序也能访问 vmalloc()返回的高端地址（它们也是由页目录、页表访问的）。

Vmlist 链表的节点类型 vm_struct 如下：

```
struct vm_struct{
unsigned long flags;          /*虚拟内存块的占用标志*/
void *addr;                   /*虚拟内存块的起始地址*/
unsigned long size;           /*虚拟内存块的长度*/
struct vm_struct *next;       /*下一个虚拟内存块*/
};
static struct vm_struct *vmlist=NULL;
```

初始时，vmlist 只有一个节点，vmlist.addr 置为 VMALLOC_START（段地址 3GB，偏移量 high_memory+8MB）。动态管理过程中，vmlist 的虚拟内存块按起始地址从小到大排序，每个虚拟内存块之后都有一个 4KB 大小的“隔离带”，用来检查指针的越界错误，用户申请大块连续空间可用 vmalloc()，如图 4-44 所示。

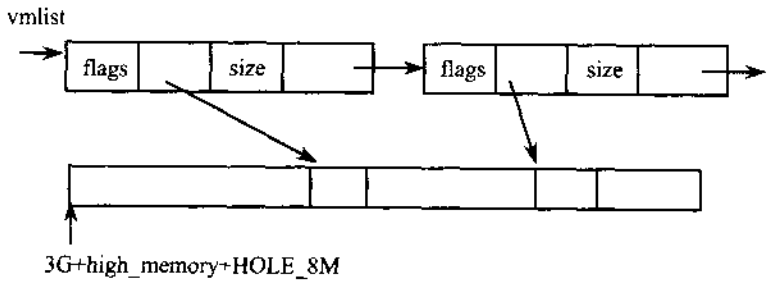


图 4-44　vmalloc()和 vfree()管理的虚拟内存

4.8.8　交换空间

计算机的物理内存空间总是影响机器性能。内存太小时，操作系统采用交换的方式。20 世纪 70 年代后，按需调页算法得到了应用，它是 Linux 操作系统采用的虚拟存储器的策略。换页操作时，Linux 区分两种不同的内存数据。一部分没有写权限的进程空间在换页时无须存入交换空间，直接丢弃即可。那些修改过的页面，换页时，其内容必须保存，保存的位置属于交换空间的某个页面。

Linux 采用两种方式保存换出的页面，一种是整个块设备，如磁盘的一个分区，称做交换设备；另一种是文件系统的固定长度的文件，称做交换文件，交换设备和交换文件统

称为交换空间。

尽管交换空间有两种不同的方式，但它们的内部格式是一致的。一个交换空间最多可容纳 32687 个页面。如果一个交换空间不够用，Linux 允许并行管理多个交换空间。交换设备远比交换文件有效。在交换设备中，属于同一页面的数据块总是连续的，第一个数据块地址确定，后续的数据块可以按顺序读出或写入。而在交换文件中，属于同一页面的数据块虽然在逻辑上是连续的，但数据块的实际位置可能是零散的。

当交换进程 kswapd 尝试换出页面时，调用测试进程 try_to_swap_out()测试页面的年龄。如果某物理页面可以换出，则调用进程 get_swap_page()申请交换空间的页面，得到交换进程的入口地址 entry，将要换出的物理页面换到 entry 指定的交换空间的某个页面中。

4.8.9 页交换进程和页面换出

当物理页面不够时，Linux 存储管理系统必须释放部分物理页面，将它们的内容写到交换空间。实现此功能的是内核态交换程序 kswapd。

Kswapd 属于一种特殊的进程，称为内核态进程。Linux 的内核态进程没有虚拟存储空间进程，它们运行在内核态，直接使用物理地址空间。它不仅能将页面换出交换空间，而且保证系统中有足够的空闲页面，保证存储系统高效地运行。

Kswapd 在系统初始启动时由内核态进程 init 创建，其初始化程序段以调用 init_swap_timer()函数结束，进而转入 while(1)循环，并马上睡眠。

内核交换进程依照 3 种方式缩减系统使用的物理页面，page cache 和 buffer cache；换出 SYSTEM V 共享内存占用的页面；换出或丢弃进程占用的页面。

1. page cache 和 buffer cache

用来保存 page cache 和 buffer cache 的页面是首选的被释放并放到 free_area 数组中的选项。用来保存存储映射的文件页面的 page cache 也可能包含了不必要的占用物理内存的页面。类似地，块设备驱动程序使用的 buffer cache 也有可能包含了不必要的缓冲区。在释放物理页面过程中，从这些 cache 中舍弃页面相对容易些，因为它不像从内存中换出页面，不需要将页面的内容写回物理设备。

当系统试图缩减这些 cache 时，它通过检测 mem_map 表中的一个表项 mem_map_t 查看是否有可以舍弃的页面。mem_map 表的表项被循环检测，每检测一个不同的表项，系统都会试图缩减描述它的内存映像，这个方法称为时钟算法。

每一个被检测的物理页面都需要核查是否是 page cache 和 buffer cache 中的一部分。共享页面此时不考虑，而且物理页面不可能同时作为两种 cache。如果一个物理页面既非 page cache，又非 buffer cache，那么检测下一个物理页面。

如果一页中所有的 buffer cache 均被释放，那么包含 buffer cache 的物理页面本身也被释放。如果被检测的物理页面是 Linux 的 page cache，那么它将从 page cache 缓冲区中释放。

当释放足够的物理页面后，kswapd 将会等待下一次被周期性唤醒。由于任何被释放的物理页面都不是任何进程的虚拟存储空间的一部分，因此不需要对任何进程页表进行修改。如果通过缩减 page cache 和 buffer cache 不能获得足够的物理页面，那么 kswapd 将试图换

出部分共享页面。

2. 换出 SYSTEM V 共享内存占用的页面

SYSTEM V 共享内存是一种进程间通信的机制，它是两个进程或多个进程之间为传递信息而共享的虚拟存储空间。

SYMTEM V 共享内存通过数据结构 shm_segs 和 shmid_ds 来描述。

```
Static struct shmid_ds *shm_segs[SHMNNI];
struct shmid_ds{ /*每一个共享内存占一个 shmid_ds 结构*/
struct ipc_perm shm_perm; /*共享内存的访问权限*/
int shm_segsz;  /*共享内存段的长度，以字节为单位*/
time_t shm_atime; /*最后一次连接时间*/
time_t shm_dtime; /*最后一次脱离连接时间*/
time_t shm_ctime;  /*最后一次更改共享关系的时间*/
unsigned short shm_cpid; /*创建共享内存段的进程的标识号*/
unsigned short shm_lpid; /*最后一次操作的进程标识号*/
short shm_nattch;      /*连接此共享内存段的进程数*/
unsigned short shm_npages; /*共享内存段占用的页面数*/
unsigned long *shm_pages; /*描述共享页帧的页表项所组成的数组*/
struct vm_area_struct *attaches;
/*每个进程均有一个 vma 描述共享内存段，attaches 指向这些 vma 段链表的表头*/
};
```

kswapd 在将 SYSTEM V 共享页面换出时，也使用时钟算法。由于一个给定的 SYSTEM V 共享内存的虚拟页面对应的物理页包含在所有共享该虚拟内存的各进程的页表中，kswapd 必须修改所有这些进程的页面，以表示页面现已不在物理内存中，而是在交换空间中。对于每个要换出的共享页面，kswapd 通过 vm_area_struct 中指向 mm_struct 类型的指针 vm_mm 找到相应各共享进程的页目录 vm_mm.pgd，进而通过 vm_offset 和 mm_struct 结构的 pgd 找到共享进程页表项。如果找到页表项有效，将它转换成“无效但被交换出”的页表项，并将用来表示共享该共享页面的进程数的变量 count 减 1。

当所有共享进程的页表项都修改完后，count 减到 0，该共享页表可以写出。同该共享内存相关的 shmid_ds 结构含有 attaches 指针（它指向的每个 vm_area_struct 结构均表示一共享进程的一段虚拟地址）。该段地址对应的页表项此时也改为换出页的页表项（指向交换空间的索引地址）。

3. 换出或丢弃进程占用的页面

如果上述两种方法不能释放出足够的页面，则交换进程就检查每一个进程，看是否有合适的页面换出。一旦被调出的进程确定换出，交换进程观察它的整个虚拟存储空间，看哪个区域未被共享或被加锁，被加锁的内存页面不能被调出。

Linux 系统并不是将进程所有可换出的页面都换出内存，而是换出一部分。

Linux 采用计龄换页算法，每页在对应的 mem_map_t 结构中都有一个年龄的属性 age，age 越小，页面越老，交换进程只换出老的进程。

如果已释放足够的可换出页面，kswapd 重新睡眠。当下一轮被唤醒时，将从当前进程的下一个进程开始换出，这样可使系统使用的页面保持平衡。

4.8.10 缺页中断和页面换入

磁盘中的可执行文件映像（Image）一旦被映射到一个进程的虚拟空间就可以开始执行。由于只有该映像区的开始部分调入内存，因此，进程迟早会执行到那些尚未调入内存的代码。当一个进程访问了一个还没有有效页表项的虚拟地址时（即页表项的 P 位为 0），处理器将产生缺页中断，通知操作系统，并将出现缺页的虚存地址（在 CR2 寄存器中）和缺页时访问虚存的模式，并传递给 Linux 的缺页中断服务程序。

系统初始化时，设定了缺页中断服务程序为 do_page_fault()：

```
Set_ttrap_gate(14, &page_faut); /*arch/I386/kernel/trap.c*/
ENTRY(page_fault)     /*arch/I386/kernel/entry.S*/
pushI$SYMBOL_NAME(do_page_fault)
jmp error_code    /*异常中断服务程序的统一入口*/
```

根据控制寄存器 CR2 传递的缺页地址，Linux 必须找到用来表示出现缺页的虚拟存储区的 vm_area_struct 结构。在搜索进程的 vm_area_struct 结构时，对搜索时间应有严格的限制。为了有效地处理搜索工作，Linux 将所有的 vm_area_struct 结构通过 AVL（Adelson-Velskii andLandis）平衡树连接起来。如果没有找到与缺页相对应的 vm_area_struct 结构，那么说明进程访问了一个非法存储区，Linux 向进程发送信号 SIGSEGV。如果进程没有处理该信号的函数，该进程将被终止。

Linux 接着检测缺页时访问模式是否合法。如果进程对该页的访问超越权限，例如试图对只允许读操作的页面进行写操作，系统也将向该进程发送一个信号，通知进程的存储访问出错。经过以上两步检查，可以确定的确是正常的缺页中断。

Linux 还区分产生缺页中断的页面是在交换空间，还是在磁盘中作为某一可执行文件映像的一部分，进而做出不同的处理，这一点通过页表项中的位来区分。如果该页面所对应的页表项是无效的（p=0）但是非空，说明缺页在交换空间中。否则，页面是某一可执行文件映像（image）的一部分。

并不是所有的 vm_area_struct 结构变量都有完整的一套虚拟存储操作，有些甚至没有 nopage 操作函数指针。在这种情况下，Linux 将使用缺省的操作函数为该虚拟页面找到一物理页帧，同时为其设置一页表项。如果 vm_area_struct 结构变量中有 nopage 操作函数，Linux 使用该操作函数。

Linux 的 nopage 函数通常用来调入已被存储映射的可执行磁盘映像，而且它是利用 page cache 来将所需映像页调入内存的。

4.8.11 存储管理系统的缓冲机制

存储管理系统的缓冲机制主要包括 kmalloc cache、swap cache 和 page cache. Kmalloc cache 在“内核态内存的申请与释放”已做了介绍，本节介绍另外两种 cache.

1. Swap cache

如果以前被调出到交换空间的页面由于进程再次被访问而调入物理内存，只要该页调

入后没有被修改过，那么它的内容与交换空间中的内容是一样的。在这种情况下，交换空间中的备份还是有效的。因此在该页再度换出时，就没有必要执行写操作。Linux 采用 swap_cache 表描述的 swap cache 来实现这种思想。Swap cache 实质上是关于页表项的一个列表，swap_cache 表位于 men_map 表之前。

每一物理页面都在 swap cache 中占有一表项，swap_cache 表项的总数就是物理页面总数。若该物理页面的内容是新创建的，或虽然曾经换出过，但换出后，该物理页面已经被修改时，则该表项清 0。内容非 0 的表项，正好是某个进程的页表项，描述了页面在交换空间中的位置。

当 Linux 将一物理页面调出到交换空间时，它先查询 swap cache，如果其中有与该页面对应的有效页表项，那就不需要将该页写出，因为原有交换空间中的内容与待换出的页面内容是一致的。

2. Page Cache

Linux 的 page cache 的作用是加快对磁盘的访问速度，文件被映射到内存中，每次读取一页，而这些页就保存于 page cache 中。

每当需要读取文件的一页时，总是首先通过 page cache 读取。如果所需页面在 page cache 中，就返回指向表示该页面的 mem-map-t 的指针。否则必须从文件系统中调入。接着，Linux 申请一物理页，将该页从磁盘文件中调入内存。

如果有可能的话，Linux 还发出读取当前页面之下一页的读操作请求。这种预读一页的思想来自局部性原理，即在进程读当前页时，它的下一页也有可能被进程用到。

随着越来越多的文件页面被读取、执行，page cache 将会变得越来越大。进程不再需要的页面应从 page cache 中删除。当 Linux 在使用内存过程中发现物理页面渐变稀少时，它将缩减 page cache。

本章小结

存储管理在操作系统中占有重要的地位，存储管理的目的是为了方便用户和提高内存利用率。存储管理的基本任务是管理内存空间、进行虚拟地址到物理地址的转换、实现内存的逻辑扩充、完成内存的共享和保护。随着计算机技术的逐步发展，存储器的种类越来越多，按照其容量、存取速度以及在操作系统中的作用，可分为 3 级存储器：高速缓存、内存和外存。

各种存储管理技术各具特点，在存储分配方式上有静态和动态、连续和非连续之分。所谓静态重定位是指在目标程序运行之前就完成了存储分配，如分区式存储管理和静态分区管理。动态重定位是指在目标程序运行过程中再实现存储分配，如动态分页管理、段式和段页式管理。连续性存储分配要求给作业分配一块地址连续的内存空间，非连续性的分配是指作业分得的内存空间可以是离散的、地址不连续的内存块，如分页式管理。在将逻辑地址转换成物理地址时，固定分区采用的是静态重定位。其他采用的是动态重定位。静态重定位是由专门设计的重定位装配程序来完成。而动态重定位是硬件地址转换机构来实

现的。虚拟存储技术是通过请求调入和置换功能，对内外进程进行统一的管理，为用户提供了似乎比实际内存容量大的多的存储器，这是一种性能优越的存储管理技术。在完成信息共享和保护方面，分区管理不能实现共享，分页管理实现共享较难，但是，分段和段页式管理就能容易实现共享。分区管理采用采用的是越界保护技术，其他的均采用越界保护和存取权控制保护相结合的方法。

在进程运行过程中，当实际内存不能满足需求时，首先淘汰最先调入内存的页面。为释放内存块给新的页面，需要进行页面置换，有很多种页面置换算法供使用。FIFO 是最容易实现的，但性能不是很好；OPT 仅具有理论价值；LRU 是 OPT 的近似算法。但是实现时要有硬件的支持和软件开销。多数页面置换算法，如最近未使用置换算法等是 LRU 的近似算法。

在大型通用系统中，往往把段页式存储管理和虚拟存储技术结合起来，形成带虚拟存储的段页式系统，它兼顾了分段存储管理在逻辑上的优点和请求式分页存储管理在存储管理方面的长处，是最通用、最灵活的系统。

本章的最后介绍了 Linux 操作系统的分页管理机制，虚存段的组织和管理，内存的共享和保护，物理空间的管理，内核态内存的申请与释放，用户态内存的申请与释放，交换空间核页面的换进和换出，缺页中断和页面调入，存储管理的缓冲机制。

习 题 四

4-1 存储管理的功能是什么？

4-2 存储分配的方式有几种？

4-3 什么叫虚拟存储器？

4-4 简述目前常用的几种分区分配方法。

4-5 简述请求页式存储管理的实现思想。

4-6 缺页中断和一般中断的主要区别。

4-7 已知页面走向为 1、2、1、3、1、2、4、2、1、3、4，且开始执行时主存中没有页面。若只给该作业分配两个物理块，当采用 FIFO 页面淘汰算法时缺页率为多少？假定现在有一种淘汰算法，该算法淘汰页面的策略为当需要淘汰页面时，就把刚使用过的页面作为淘汰对象，试问相同的页面走向，其缺页率又为多少？

4-8 4-8 题图给出了某系统中的空闲分区表，系统采用可变式分区存储管理策略。现在以下列作业序列：96k、20k、200k。若用首次适应算法和最佳适应算法来处理这些作业序列，试问哪一种算法可以满足该作业序列的请求，为什么？

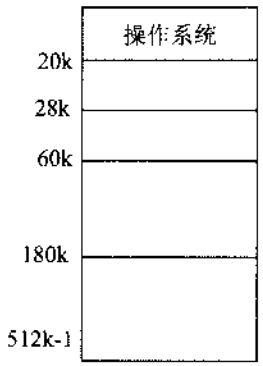
4-9 在系统中，采用固定分区分配管理方式，内存分区的情况如 4-9 题图所示。现在有大小为 1K、9K、33K、121K 的多个作业要求进入内存，设内存分配策略采用最佳分配算法，试画出它们进入到内存后内存空间的分配情况，并说明内存浪费有多大？

4-10 有一个分页存储管理系统，页面大小为每页 100 字节。有一个 50×50 的整型数组，按行连续存放，每个整数占两个字节，将数组初始化为 0 的程序描述如下：

空闲分区

分区号	大小	起始地址
1	32k	100k
2	10k	150k
3	5k	200k
4	218k	220k
5	96k	530k

4-8 题图



某系统内存分区情况

4-9 题图

```
    int a[50][50];
    int i,j;
for (i=0;i<=49;i++)
for (j=0;j<=49;j++)
a[i][j]=0;
```

若在程序执行时内存中只有一个存储块来存放数组信息，试问该程序执行时产生多少次缺页中断？

4-11　试述分页系统和分段系统的主要区别。

4-12　设有一页式存储管理系统，向用户提供的逻辑地址空间最大为 16 页，每页大小为 2KB，内存总共有 8 个存储块，试问逻辑地址至少应为多少位？内存空间多大？

4-13　有一段式存储管理系统，其段表为：

段号	段内位移
0	430B
1	10B
2	500B
3	400B
4	112B
5	32B

已知各操作数的逻辑地址为(0,400)，(1,80)，(4,150)，(5,20)，试求它们所对应的物理地址。

4-14　在一个请求分页存储管理系统中，一个作业的页面走向为 4、3、2、1、4、3、5、4、3、2、1、5，当分配给该作业的物理块数分别为 3、4 时，试计算采用下述页面淘汰算法时的缺页率，并比较所得的结果。

（1）最佳置换淘汰算法。

（2）先进先出淘汰算法。

（3）最近最久未使用淘汰算法。

4-15 在采用页式存储管理系统中，某作业 J 的逻辑地址空间为 4 页（每页 2KB），且已知该作业的页面映像表如下：

页号	块号
0	2
1	4
2	6
3	8

试求有效逻辑地址 4865B 所对应的物理地址。

第5章　文件管理

本章学习目标

在使用计算机的过程中，要用到大量程序和数据，且要反复使用，所以必须长期保存这些信息。大量的备用程序和数据存放在外存，当需要运行或处理时，将其调入内存。

从用户逻辑出发，信息分为程序和数据。用户关心的是程序、数据整体。用户要给备部分独立信息命名，并通过名字来使用这些信息，我们称各部分独立的信息为文件。

- 通过本章的学习，读者应该掌握以下内容:
- 掌握文件的目录结构
- 掌握文件的按名存取方式
- 掌握操作系统对文件和目录的主要操作过程
- 掌握 Linux 系统中文件的使用方法
- 掌握操作系统对文件存储空间的管理方法

5.1　文件管理概述

5.1.1　文件及其分类

1. 文件的概念

文件的概念是在信息的物理存储和信息表示方式的基础上引入的。一般定义文件是具有符号名而且在逻辑上具有完整意义的信息项的有序序列。讨论文件时经常用到以下术语。

（1）数据项。数据项是描述一个对象的某种属性的字符集，是数据组织中可以命名的最小逻辑数据单位，即原子数据，又称为数据元素或字段。它的命名往往与其属性一致。例如，用于描述一个教职工的基本数据项有姓名、年龄、工资等。

（2）记录。记录是一组相关数据项的集合，用于描述一个对象在某方面的属性。例如，一个学生记录有学号、姓名、性别、年龄、班级等。

文件是相关记录的集合，它通常存放在外存上，可以作为一个独立的单位存放和实施相应的操作。例如，用户编写的一个源程序、经编译生成的目标代码程序、系统中的库程序和各种系统程序、一批待加工处理的数据、一篇文章等，都可构成一个文件加以保存。

文件必须有文件名，通常由一串 ASCII 字符或（和）汉字构成，名字的长度因系统不同而异。Linux 系统中，文件名最大长度由 NR-NAME-LEN 控制，默认值为 255 个字符。

文件的属性包括文件类型、文件长度、文件的物理位置、文件的存取权限、文件的建立及修改日期等。

2. 文件类型

为便于文件的控制和管理，通常把文件分成若干类型，但由于不同系统对文件的管理方式不同，因而对文件的分类方法有很大差异。下面介绍几种常用的分类方法。

（1）按文件的数据形式分类。

1）源文件：由源程序和数据构成的文件，一般由 ASCII 字符或汉字组成。

2）目标文件：由相应的编译程序编译而成的文件，由二进制组成，扩展名为.Obj。

3）可执行文件：由目标文件连接而成的文件，扩展名一般为.exe。

（2）按用途分类。

1）系统文件：由系统进行管理及为用户提供基本服务的文件，如操作系统、编译系统、编辑系统、预处理系统等。这类文件对用户不直接开放，只能通过系统调用为用户服务。

2）库文件：由标准子程序及常用的应用程序组成的文件。这类文件允许用户调用，但不允许用户修改。

3）用户文件：用户直接使用的文件，并由文件拥有者进行读/写和执行的文件，如源程序文件、目标程序文件以及由原始数据、计算结果等组成的文件。

（3）按存取权限分类。

1）只读文件：允许授权用户读，但不准改写文件内容。

2）读写文件：允许授权用户读、写，但禁止未授权用户读、写的文件。

3）可执行文件：允许授权用户执行，但通常也允许读或写。

（4）按保存时间分类。

1）临时文件：用户在一次解题过程中建立的中间文件，当用户撤离系统时，该文件往往也随之被撤消。

2）档案文件：只保存在作为档案的磁带上，以便考证和恢复用的文件，如日志文件。

3）永久文件：长期保存的有价值的文件，以备用户经常使用的文件。它不仅在磁盘上有文件副本，而且在“档案”上也有一个可靠的副本。

（5）在 UNIX 和 DOS 操作系统中，文件按内部结构和处理方式分类。

1）普通文件：由表示程序、数据或正文的字符串构成的文件，内部没有固定的结构。这种文件既可以是系统文件，也可以是库文件或用户文件。

2）目录文件：由文件目录构成的一类文件。对它的处理（读、写、执行）在形式上与普通文件相同。

3）特别文件：特指各种外部设备，为了便于管理，在 UNIX 和 DOS 中，把所有输入/输出设备都按文件格式供用户使用。这类文件对于查找目录、存取权限验证等的处理与普通文件相似，而其他部分的处理要针对设备特性要求做相应的特殊处理。

应该指出，采取不同的分类方式将导致不同的文件系统。

5.1.2　文件系统及其功能

现代操作系统中都配置有较完备的文件管理系统，简称文件系统。所谓文件系统，是指操作系统中实现对文件的组织、管理和存取的一组系统程序，它实现对文件的共享和保护，方便用户按名存取。

1. 文件系统的结构

文件系统的结构模型如图 5-1 所示。

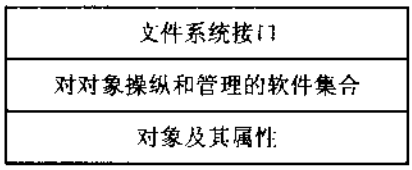


图 5-1　文件系统模型

（1）对象及其属性。文件管理系统管理的对象有：

1）文件。它作为文件管理的直接对象。

2）目录。为了方便用户对文件的存取和检索，在文件系统中必须配置目录。对目录的组织和管理是方便用户和提高对文件存取速度的关键。

3）磁盘（磁带）存储空间。文件和目录必定占用存储空间，对这部分空间的有效管理不仅能提高外存的利用率，而且能提高对文件的存取速度。

（2）对对象操纵和管理的软件集合。这是文件管理系统的核心部分，其中包括对文件存储空间的管理、对文件目录的管理、用于将文件的逻辑地址转换为物理地址的机制、对文件读和写的管理，以及对文件的共享与保护等功能。

（3）文件系统的接口。为方便用户使用文件系统，文件系统通常向用户提供两种类型的接口：

1）命令接口。指用户与文件系统交互的接口。用户可通过键盘终端键入命令，取得文件系统的服务。

2）程序接口。指作为用户程序与文件系统的接口。用户程序可通过系统调用来取得文件系统的服务。

2. 文件系统的功能

（1）文件及目录的管理，如打开、关闭、读、写等。

（2）提供有关文件自身的服务，如文件共享机制、文件的安全性等。

（3）文件存储空间的管理，如分配和释放，主要针对可改写的外存，如磁盘。

（4）提供用户接口。方便用户使用文件系统所提供的服务，称为接口。文件系统通常向用户提供两种类型的接口：命令接口和程序接口。不同的操作系统提供不同类型的接口，不同的应用程序往往使用不同的接口。

5.2 文件的结构

文件的结构是指文件的组织形式。任何一个文件都存在两种形式的结构。

（1）文件的逻辑结构。从用户观察和使用文件的角度出发所定义的文件组织形式。

（2）文件的物理结构。从系统的角度考察文件在实际存储设备上的存放形式，又称为文件的存储结构。

5.2.1 文件的逻辑结构

从用户对文件元素的操作出发，建立文件的逻辑结构。文件逻辑结构决定对文件构成元素的操作方法（算法），如查找一个特定元素。逻辑结构是用户可见的，是可以直接处理的，独立于文件的物理结构。

按文件的逻辑结构，将文件分为有结构文件和无结构文件。

1. 有结构文件

有结构文件又称为记录式文件，它把文件内的信息划分为多个记录，用户以记录为单位组织信息，即在逻辑上可被看成是一组连续顺序的记录的集合。每个记录是一组相关的数据项集合，每个数据项用于描述一个对象某个方面的属性，如姓名、年龄、性别、工资等。

有结构文件按其记录的长度是否相同可分为定长记录文件和不定长记录文件两种。

（1）定长记录文件是指文件中所有记录的长度都相同。文件的长度可用记录的数目来表示。

（2）不定长记录文件是指文件中几个记录的长度不相同，如姓名、家庭住址、性别等，有长有短。在处理之前每个记录的长度是已知的。

2. 无结构文件

无结构文件又称字符流式文件，是由字符序列组成的文件，其文件内部不再划分记录，文件长度直接按字节来计算，如大量的源程序、可执行文件、库函数等都是无结构文件形式。在 Linux 系统中，所有文件都被看作是流式文件，系统不对文件进行格式处理。

3. 文件存取方式

文件存取方式是指用户对文件的逻辑存取方式，由文件的性质和用户使用文件的情况决定。常用的存取方式有顺序存取和随机存取。

（1）顺序存取方式。按照文件的逻辑地址依次存取。记录式文件是按照记录的排序顺序依次存取，如当前存取的记录为 R_i，则下次要存取的记录自动被确定为 R_{i+1}。在只提供无结构的流式文件系统中，顺序存取法按读写位移（offset）从当前位置开始读写，即每读写完一段信息后，读写位移自动加上这段的长度，然后再根据该位移读写下面的信息。

（2）随机存取方式。它基于磁盘的文件模式，又称为直接存取方式，允许用户随意存取文件中的任何一个物理记录，而不管上次存取了哪一个记录。在无结构的文件中，随机存取法必须事先用必要的命令把读写位移到要进行读写的信息开始处，然后再进行读写。

5.2.2　文件的物理结构

文件的物理结构是指文件在外部存储器上的存取方式，以及它与文件逻辑结构之间的对应关系，即文件的存储结构。

在外存上如何存放文件主要与下述两个因素有关：

（1）检索速度。不同的文件物理结构将产生不同的检索速度。

（2）存储介质。选用哪种存储介质与用户对文件存取方式的要求有关。

通常，文件的物理结构有连续文件结构、串联文件结构、索引文件结构等。

1. 连续文件

一个文件顺序存放在外存的若干个连续物理块中，称这种文件为连续文件。连续文件保证了逻辑文件中的记录顺序与存储器中文件占用盘块的顺序是一致的。连续文件的文件控制块 FCB（File Control Block，）是用于描述和控制文件的数据结构。在连续文件的文件控制块中记录着文件所占用的起始物理块号和物理块数。如图 5-2 所示为连续文件的结构，假定记录和盘块的大小相同。

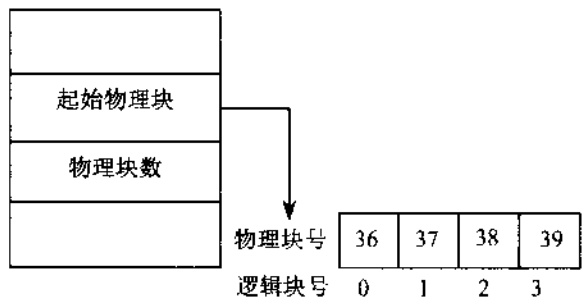


图 5-2　连续文件结构

连续文件的主要优点有顺序访问容易，访问一个占有连续空间的文件，非常容易，同时连续分配也支持直接存取；顺序访问速度快，因为由连续分配装入的文件所占用的盘块可能是位于一条或几条相邻的磁道上，所以磁头的移动距离最少，访问速度快，常用于存放系统文件等固定长度的文件，如编译程序文件、操作系统文件和由系统提供的实用程序文件等。

但连续文件也存在以下缺点：要求建立文件时就确定它的长度，依此来分配存储空间，往往难以实现；不利于文件长度的动态增加；反复删除记录后，易产生碎片，导致外存空间利用率低。

2. 串联文件

把一个逻辑上连续的文件离散地存放在不连续的物理块中，为了表示其对应的逻辑块次序，对各物理块设置一个指针（称为连接字），指向下一个逻辑块对应的物理块，从而使存放同一文件的物理块链接成一个串联队列，这样形成的物理文件称为串联文件，又称为链接文件。串联文件的优点是支持离散分配，因而消除了碎片，存储空间的利用率提高了；能实现按需分配且无需事先知道文件长度，支持文件的动态增长，并方便对文件增、

删、改，它具有以下两种链接结构。

（1）隐式链接。把一个逻辑文件分为若干个逻辑块，每个块的大小与物理块的大小相同，并为逻辑块取从 1～n 的编号，再把每一个逻辑块存放到对应的物理块中。在每个物理块中设置一个链接指针，通过这些指针把存放了该文件的物理盘块连接起来，由于连接指针只存放在文件的物理块中，所以称为隐式链接，其结构如图 5-3 所示。

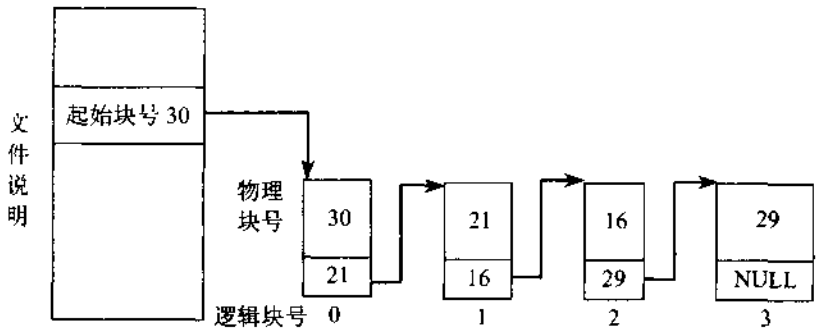


图 5-3 隐式链接结构

这种结构的优点是克服了连续文件的缺点，但是又带来了新的问题，如只适合顺序访问，对随机存取极其低效；同时，由于仅通过链接指针来实现各离散盘块的链接，所以只要其中任何一个指针出现问题，都会导致整条链的断开，因而可靠性较差。为了提高检索速度，可将几个盘块组成一个簇，以簇为单位进行盘块分配，但又会带来内部碎片增大的缺点。

（2）显式链接。把用于链接文件各物理块的指针显式地放在一张链接表中，该表在整个磁盘上仅设置一张，如图 5-4 所示，表的序号是物理块号。在每个表项中存放链接指针，指向属于同一文件的下一物理块。文件的第一个物理块号存放在其 FCB 中。由于此表展示了文件在外存上的存储情况，所以称此表为文件分配表 FAT（File Allocation Table）。FAT 同时又展示了外存中空闲物理块的情况。FAT 实际就是文件的一张显示链接表。MS-DOS、Windows 都采用了 FAT 这种文件物理结构（文件存储方式）。

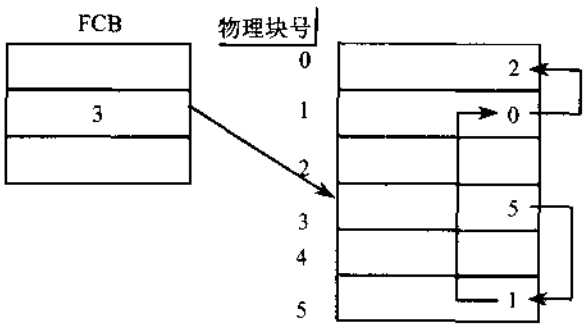


图 5-4 显式链接结构

优点：相对于隐式链接而言，易于采用随机存取，即随机存取速度快，因为只需将 FAT 一次性装入内存，就可查找所有文件物理块。

如图 5-5 所示为 MS-DOS 的文件物理结构。图中有两个文件，文件 F1 占 3 个盘块，块号为 8、4、7；文件 F2 占用 5、10、9 三个盘块。每个文件的第一个盘块号应放在该文件的文件控制块中。整个系统一张文件分配表 FAT，在 FAT 的每个表项中存放下一个盘块号。对于一个 1.2M 的软盘，盘块大小为 1KB，共有盘块 12000 个，每个分配表项占 16 位，即 2B，所以共需 24KB 的存储空间。对于 540MB 的硬盘，共有盘块 540MB/1KB=540K∈(219,220)，故文件分配表表项应取 20 位，即 2.5B，所以其文件分配表需占用存储空间为 540K×2.5B=1350KB。

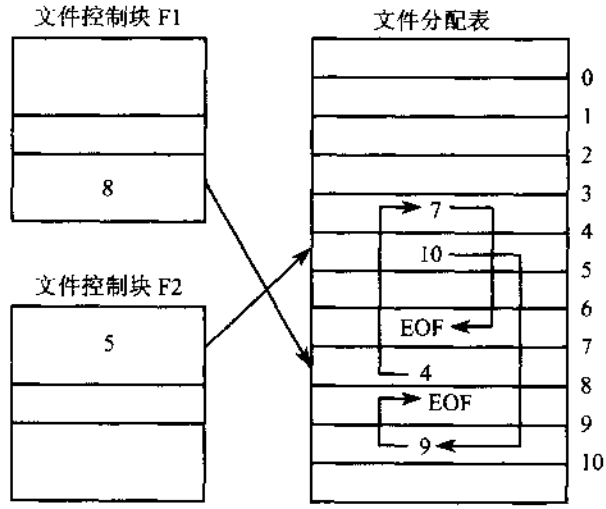


图 5-5　MS-DOS 的文件物理结构

链接分配文件方式虽然解决了连续文件方式存在的问题，但又出现了另外两个问题：

（1）不能支持高效的直接存取，因为若对一个较大的文件进行直接存取，必须首先在文件分配表中顺序地查找许多盘块号。

（2）文件分配表需占用较大的内存空间。

3. 索引文件

索引文件是实现非连续分配的另一种方案，系统为每个文件建立一张索引表，索引表是文件逻辑块号和物理块号的对照表。此外，在文件控制块中设置了索引表指针，它指向索引表的始址，索引表存放在盘块中，当索引表较大，需占用多个盘块时，可通过连接指针将这些盘块链接起来，如图 5-6 所示。

索引文件克服了连续文件和串联文件的不足，既能方便迅速地实现随机存取，又能满足文件动态增长的需要。由于它的检索速度较快，所以主要用于对信息处理及时性要求较高的场合，但是增加了索引表带来的存储空间开销。在存取文件时，需要先取出索引表，然后再查表，得到物理块号，这样增加了存取时对存储器的访问次数，降低了文件的存取速度，加重了输入输出的负担。一种改进办法是将索引表部分或全部放入内存，以内存空间为代价换取存取速度的改善。

4. 多重索引文件

若文件很大，那么不仅存放文件信息需要大量盘块，而且相应的索引表也必然很大。

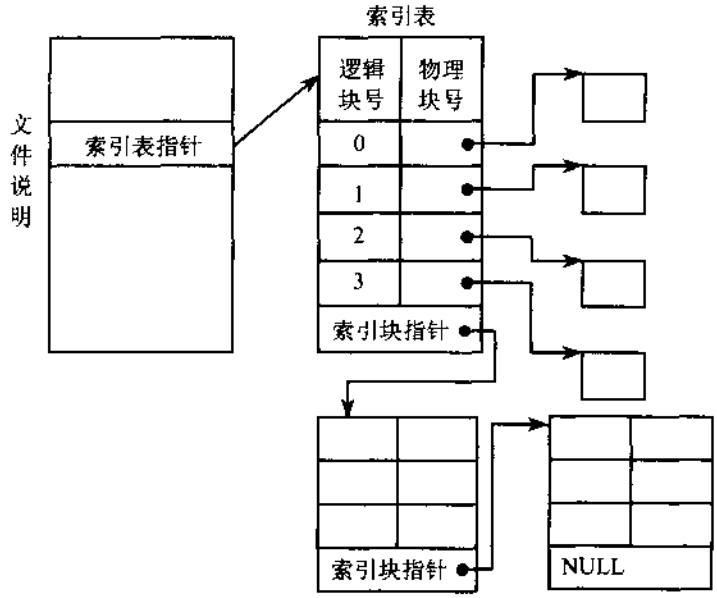


图 5-6 索引文件结构

例如，若盘块大小为 1KB，那么长度为 1000KB 的文件就需要 1000 个盘块，索引表项要有 1000 项。若盘块号用 4 个字节表示，则索引表至少占用 4000KB。显然，把索引表放入内存是不合适的，而且不同文件其大小也不同，文件在使用过程中很可能需要扩充空间。采用单重索引结构无法满足灵活性和节省内存的要求，为此引入了多重索引结构，其结构如图 5-7 所示。在这种结构中采用了间接索引方式，即由最初索引项中得到某一盘块号，该块中存放的是另一组盘块号，后者每一盘块中又可存放下一组盘块号，最后的盘块中存放的一定是文件内容。

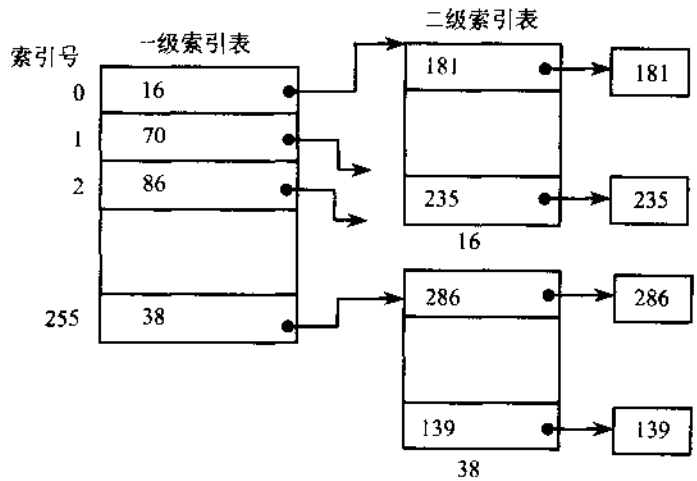


图 5-7 多重索引文件结构

多级索引具有索引文件的优点，但也会带来因为多次访问磁盘而影响速度的缺点。

5.3　文件目录管理

文件目录是查看、读取外存中所存放文件而采用的数据结构，用于文件描述和文件控制，实现按名存取和文件共享与保护，随文件的建立而创建，随文件的删除而消亡。在很多操作系统中，对目录采用文件的方式进行管理。

5.3.1　文件控制块和索引节点

1. 文件控制块

为了便于对文件进行控制和管理，必须为文件设置用于描述和控制文件的数据结构，这种数据结构称为文件控制块（FCB），文件与文件控制块一一对应。文件控制块的有序集合称为文件目录，即一个文件控制块就是一个文件目录项。完全由目录项构成的文件称为目录文件。文件控制块通常由文件的基本信息、存取控制信息、文件使用信息组成。

（1）基本信息。

文件名：用于标识一个文件的名字，每个文件必须有惟一的文件名，用户可根据文件名对文件进行操作。

文件类型：指明文件属性是系统文件还是用户文件，是普通文件还是目录文件，或是特别文件等。

文件物理位置：指明文件在外存上存放的物理位置和范围，包括文件的设备名、文件在外存的起始地址、文件长度等。

文件的逻辑结构：指明文件是记录式文件还是字符流式文件，若是记录式文件，需指明记录个数及记录是变长记录还是定长记录等。

文件的物理结构：指明文件是连续文件、串联文件或是索引文件。

（2）存取控制信息。

文件所有者（属主）：通常是创建文件的用户，或者改变已有文件的属主。

访问权限（控制各用户可使用的访问方式）：为了防止用户有意或无意地破坏文件，对文件设立的保护，规定其允许访问的操作类型，如读、写、执行、删除等。

（3）使用信息。

日期和时间：文件创建和上一次修改的日期和时间。

当前使用的信息：包括有多少个进程在使用该文件，是否被其他进程锁住等信息。

应该说明，对于不同 OS 的文件系统，有不同的文件目录，也就是说，不同的 OS 其文件控制块中所包含的信息也不同，可能只含有上述信息中的某些部分。

MS-DOS 系统中的文件控制块的长度为 32 个字节，它含有文件名及文件扩展名共 11 个字节、文件所在的首块号、文件属性、文件大小及文件建立和修改的日期等。利用文件所在的首块号作为物理块链接表的索引，按索引链向下查找，可找到文件所在的盘块。如图 5-8 所示是 MS-DOS 的文件控制块示意图。

文件名	扩展名	属性	保留	时间	日期	首块号	大小

图 5-8 MS-DOS 的文件控制块

2. 索引节点

（1）索引节点的引入。我们通常把文件目录存放在磁盘上，当文件很多时，文件目录可能需要占用大量的盘块。在查找目录的过程中，先将目录文件中第一个盘块中的 FCB 调入内存，然后把用户给定的文件名与 FCB 中的文件名进行比较，若未找到，则将下一个盘块调入内存，直到找到或确定没有与之匹配的文件名为止。设文件目录占用的盘块数为 M，按此方法查找，则查找一个目录项平均需调入盘块（M+1）/2 次。例如，一个 FCB 长度为 128B，盘块大小为 1KB，则一个盘块只能存放 8 个 FCB，若一个文件目录共有 640 个 FCB，则需占用 80 个盘块，平均查找一个文件需要启动磁盘 40 次。

经过分析可以发现，在检索文件目录的过程中，只用到了文件名，只有目录项中的文件名与指定的文件名相匹配时，才需从 FCB 中读出该文件的物理地址，而其他一些描述信息在检索目录时一概不用，显然，这些信息在检索目录时不需要调入内存。为此，在有些系统中，便采用了文件名与文件描述信息分开的办法，即把文件描述信息单独形成一个称为索引节点的数据结构，简称为 i 节点。文件目录中的每个目录项仅由文件名及指向该文件所对应的索引节点的指针构成，如 Linux 系统中的目录项只由文件名和 i 节点号组成，如图 5-9 所示。

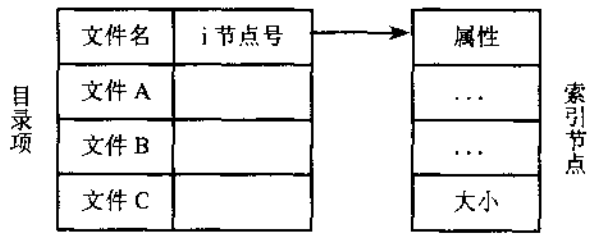


图 5-9 Linux 系统的文件目录

Linux 系统中，一个目录项共占 16B，其中文件名占用 14B，i 节点占用 2B。在大小为 1KB 的盘块中，可存放 64 个 FCB。在一个共有 640 个 FCB 的文件目录中查找一个文件时，平均只需启动磁盘 5 次，因此大大减少了系统开销。

（2）磁盘索引节点，是指存放在磁盘上的索引节点。每个文件有惟一的一个磁盘索引节点，它主要包括以下内容：

文件所有者标识号：指拥有该文件的文件主或同组的标识符。

文件类型：指明文件是普通文件、目录文件还是特别文件等类型。

文件物理地址：指出数据文件所在的物理块号。如在 Linux 系统中，通过 15 个地址项来表明文件所在的物理块号。

文件存取权限：用户对文件的操作类型，如读、修改、执行等。

文件大小：文件所占有的字节个数。

文件连接计数：指明系统中共享该文件的进程个数。

文件存取时间：指出该文件最近被进程存取的时间、最近被修改的时间及索引节点最近被修改的时间等。

（3）内存索引节点，是指存放在内存的索引节点。当文件打开时，要将磁盘索引节点复制到内存索引节点中，便于以后使用。内存索引节点包括以下内容：

索引节点编号：标识内存索引节点。

索引节点状态：指示该节点是否已被修改或已被上锁。

访问计数：当进程访问该节点时，访问计数加 1，访问完再减 1。

链接指针：指向空闲链表和散列队列的指针。

逻辑设备名：含有该文件的文件系统的逻辑设备名。

5.3.2　文件目录结构

文件目录结构的组织关系到文件的存取速度、文件的共享性和安全性就如同一个企事业单位内部行政机构的设置。因此，组织好文件的目录是设计文件系统的重要环节。下面介绍几种常用的目录结构组织形式。

1. 一级目录结构

一级目录结构是最简单的目录结构，如设备目录就是一级目录，它是指把系统中的所有文件都建立在一个目录下，每个文件占用其中的一个目录项，每个目录项中包含了文件名、文件物理地址、文件说明信息等，如图 5-10 所示。

	文件名	物理地址	文件说明	状态位
目录项	文件名 1			
	文件名 2			
	…			

图 5-10　一级文件目录结构

一级目录结构主要用于单用户操作系统，它具有如下优点：

（1）结构简单，通过管理其目录文件，便可实现文件信息的管理。

（2）实现按名存取。

同时，一级目录结构具有以下缺点：

（1）文件较多时，目录检索时间长。

（2）有命名冲突。简单的文件目录结构中，文件名和文件实体之间存在着一一对应关系，即不允许两个文件具有相同的名字。在多道程序系统中，尤其是多用户的分时系统中，重名很难避免，这就很难准确地找到用户需要的文件。显然，如果人工管理文件名注册，以避免命名冲突非常麻烦。

（3）不便于实现文件共享。一级目录结构要求所有用户用相同的名字访问同一个文件。

2. 二级目录结构

在多用户系统中，为解决不同用户的文件重名问题，可以建立两级目录结构。在两级目录结构中，除了系统目录外，还在系统目录下为每个用户建立一个用户文件目录（第二级目录）；在用户目录下是该用户的文件，而不再有下级目录。二级目录结构适用于多用户系统，各用户可有自己的专用目录，如图 5-11 所示。

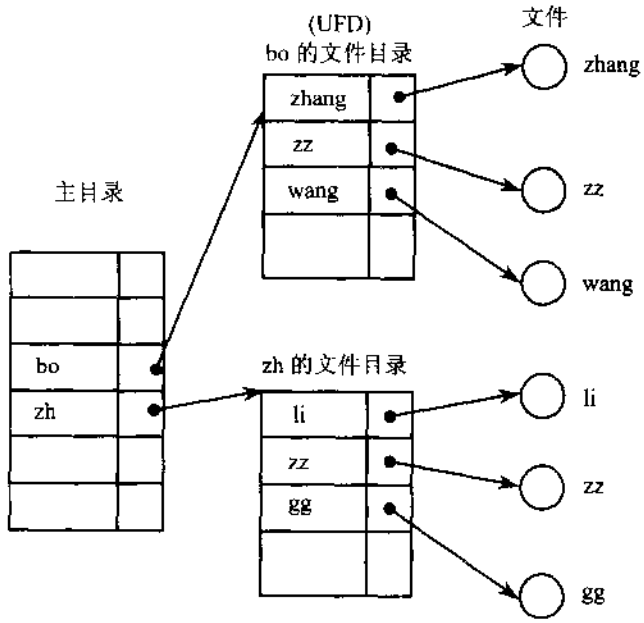


图 5-11 二级目录结构

二级目录结构基本上克服了一级目录结构的缺点而具有以下优点：

（1）提高了检索目录的速度。

（2）在不同的用户目录中可以使用相同的文件名。

（3）不同用户可使用不同的文件名来访问系统中的同一个共享文件。

3. 多级目录结构

（1）目录结构。为了给某些使用多个文件的用户提供检索方便，以及更好地反映实际上多层次的复杂的文件结构关系，可以把二级目录自然推广到多级目录，即树形目录。如图 5-12 所示为多级目录结构。

（2）路径。路径是指从树型目录中的某个目录层次到某个文件的一条道路。此路径的主要构成是目录名称，中间用“/”或“\”分开。任意文件在文件系统中的位置都是由相应的路径决定的。用户对文件进行访问时，要给出文件所在的路径。路径一般分为相对路径和绝对路径。

注意：绝对路径名以“/”打头。

相对路径名常和工作目录（也称当前目录）一起使用。用户可以指定一个目录作为当前的工作目录。这时，所有的路径名如果不是从根目录开始，则都是相对于工作目录的。

在图 5-12 中，若当前目录为/A，则文件 L 的相对路径为 AC/L。

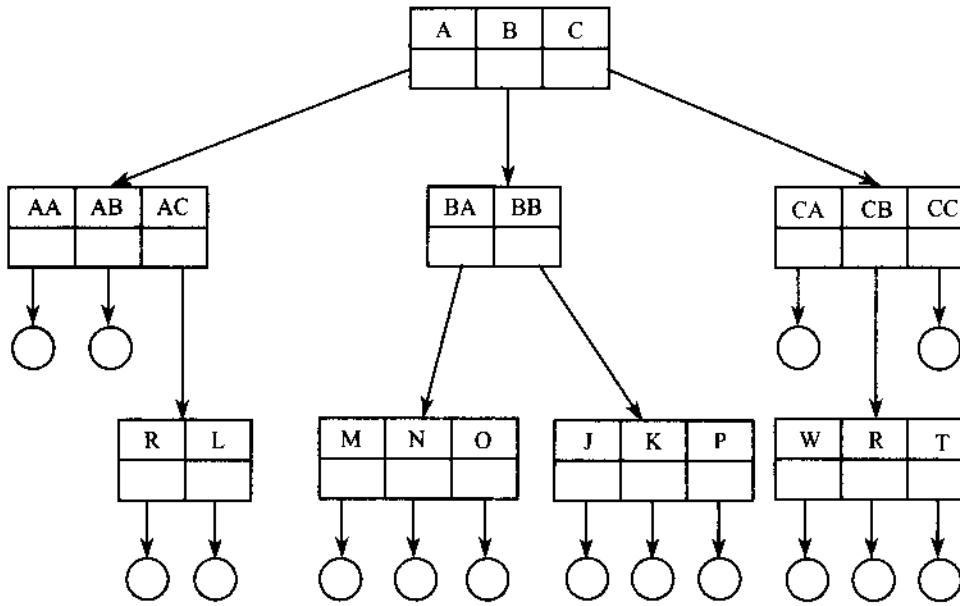


图 5-12　多级目录结构

注意：相对路径名是从当前目录的下级开始书写。

说明：大多数支持树型结构的操作系统，在每个目录中有两个特殊的目录项“ • ”和“ •• ”，通常读作“点”和“点点”。“点”指当前目录，“点点”指其父目录。在图 5-12 中，若某进程的工作目录为/A/AC/R，它可以使用“ •• ”沿树向上到达其父目录/A。

多级目录结构具有以下几个优点：

（1）能有效地提高对目录的检索速度。

（2）允许文件重名。允许用户在自己的分目录中使用与其他用户相同的文件名。

（3）便于实现文件共享。一是允许不同的用户按自己的命名习惯为共享的文件赋予不同的名字，即不同的用户可以利用不同的文件名访问同一个共享文件。二是容易实现文件共享，若一用户欲共享另一用户的文件，只需在自己的目录文件中增设一表目，其中的文件名项使用自己赋予该文件的符号名字，并填上该共享文件的惟一标识符即可。

5.3.3　按名存取

用户访问文件时，系统首先根据文件名查找文件目录，找到它的文件控制块或索引接点号；其次，经过合法性检查，从控制块或索引节点中找到该文件所在的物理地址，换算为物理位置；然后，启动磁盘驱动程序，将所需的文件读入内存，进行相应的操作。目前，对文件目录进行查找的方式有顺序检索法和 Hash 方法。

1. 顺序检索法

顺序检索法又称线性检索法，在一级目录结构中，利用用户提供的文件名，用顺序查找

的方法直接从文件目录表中找到指定文件的目录项。在树形目录结构中，用户提供的文件名是由多个文件分量名组成的路径名，此时需对多级目录进行查找，即系统先读入第一个文件分量名，用它和根目录文件或当前目录文件中各个目录项进行比较，若找到匹配者，便可找到匹配项的文件控制块或索引节点，然后再读入路径名中的第二个分量名，用它和相应的第二级文件目录中各个文件目录项的文件名顺序比较，若找到匹配项，再取第三个、第四个文件分量名进行比较，直至全部查完，最后可得到数据文件的文件控制块或索引节点。若在查找过程中发现一个分量名没有查找到，则应停止查找并返回“文件未找到”的信息。

2. Hash 方法

建立一张 Hash 索引的文件目录，利用 Hash 方法进行查找，即系统利用用户提供的文件名，将它变为文件目录的索引值，再利用该索引值到目录中去查找，提高平均检索速度。顺便指出，现代操作系统通常提供模式匹配功能，即在文件名中使用通配符“*”、“？”等。对于使用通配符的文件名，系统无法利用 Hash 方法进行检索目录。因此，还是需要利用顺序检索法来查找目录。

5.3.4 Linux 文件目录

Linux 系统以文件目录的方式组织和管理系统中的所有文件。文件目录将所有文件的说明信息采用树形结构组织起来，即我们常说的目录，整个文件系统有一个“根”（Root），然后在根上分“杈”（Directory），任何一个分杈上都可以再分杈，杈上也可以长出“叶子”。“根”和“杈”在 Linux 中被称为是“目录”或“文件夹”，而“叶子”则是一个个的文件。

Linux 系统通过目录将系统中所有的文件分级、分层组织在一起，形成了 Linux 文件系统的树形目录结构。以根目录为起点，其他目录都由根目录派生而来。一个典型的 Linux 系统树形目录结构如图 5-13 所示，用户可以浏览整个系统，进入任何一个已授权进入的目录，并访问那里的文件。

Linux 目录提供了管理文件的方便途径。每个目录里面都包含文件。用户可以为自己的文件创建自己的目录，也可以把一个目录下的文件移动或复制到另一目录下，而且能移动整个目录，并和系统中的其他用户共享目录和文件，也就是说，能够方便地从一个目录切换到另一个目录，而且可以设置目录和文件的管理权限，以便允许或拒绝其他人对其进行访问。文件目录结构的相互关联性使共享数据变得十分容易，几个用户可以访问同一个文件，因此允许用户设置文件的共享程度。

需要说明的是，根目录是 Linux 系统中的特殊目录。Linux 是一个多用户系统，操作系统本身的驻留程序存放在以根目录开始的专用目录中，有时被指定为系统目录，图 5-13 中那些根目录下的目录就是系统目录。

如前所述，目录是 Linux 系统组织文件的一种特殊文件。为使用户更好地使用目录，我们介绍有关目录的一些基本概念。

1. 工作目录与用户主目录

从逻辑上讲，登录到 Linux 系统后，每时每刻都处在某个目录之中，此目录称作工作

目录或当前目录（Working Directory）。工作目录可以随时改变。用户初始登录到系统中时，其主目录（Home Directory）称为工作目录。工作目录用“.”表示，其父目录用“..”表示。

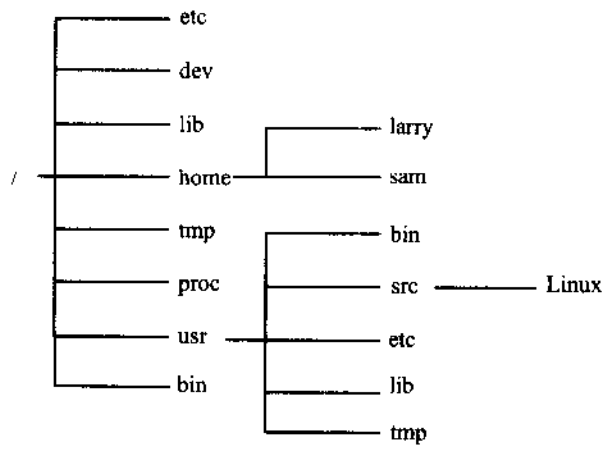


图 5-13　Linux 系统的树形文件目录结构

用户主目录是系统管理员增加用户时建立的（以后也可以改变），每个用户都有自己的主目录，不同用户的主目录一般互不相同。用户刚登录到系统时，其工作目录便是该用户主目录，通常与用户的登录名相同。用户可以通过一个“~”字符来引用自己的主目录。例如，当前用户主目录为/home/zhang，则$ cat ~/class/software_1 和$ cat /home/WANG/class/software_1 意义相同，将用用户主目录名来替换“~”字符。目录层次建立好后，就可以把有关的文件放到相应的目录中，从而实现对文件的组织。

2. 根目录（/）

目录结构上的最高点称为根目录，它使用与超级用户相同的名称。单个字符斜杠（“/”）表示根目录。

Linux 系统中的其他目录都包含在根目录之下的层次结构中，这一点不同于 Windows 系统，Windows 系统中的每个驱动器被赋予了自己的字母及其自己的目录结构。在 Linux 中，系统上所有的存储设备都被装载到根目录之下的每个目录中，或者直接在根目录下，或者更下一层。

注意：/目录与 root 用户的主目录不是一回事，其主目录为/root，因此/root 目录是/的子目录。

3. Linux 子目录

（1）/bin。这个目录包含超级用户和一般用户使用的命令，这些命令提供一些操作，如复制、移动和删除文件，登录、创建和打开文件，识别系统名称，查看文本文件等。用户通常不会去改变/bin 目录的内容。

（2）/dev。/dev 目录包含设备文件和其他特殊文件。

（3）/etc。这个目录包含启动和正常运行 Linux 系统所需的配置文件，这些文件大多

能够被编辑（通过配置工具或文本编辑器来完成）。大多数 Linux 集成套件提供了许多辅助软件用于配置/etc 目录中的文件，以便使用户更容易地使用 Linux。在安装过程中用户所回答的一些问题将自动填充到相关的/etc 目录文件中。

（4）/home。在典型情况下，这个目录拥有系统中每个用户的子目录。例如，如果 Mom、Dad、Erin 和 Matt 是系统中的所有用户，那么/home 目录可以包含 4 个用户目录：

/dad

/erin

/matt

/mom

如果系统中有大量用户，可以将它们分组放入部门子目录。有的 Linux 系统根本不使用/home 目录，并且将主目录放置在其他地方，但这种系统比较少见。

（5）/boot。这个目录包含系统启动所需的大多数文件，电脑启动时需要的其他文件存储在/etc 和/shin 目录中。

（6）/lib。这个目录包含了位于/bin 目录和/shin 目录中程序需要的库文件。一个库文件是一个程序文件，它包含能够被多个不同程序使用的代码。将这些共用代码以库的形式存放起来，可以减轻程序设计者的工作量。

（7）/proc。这个目录用于同 Linux 内核交换数据。该目录中有一些能够查看的文本文件，它们包含一些系统信息，如内核版本、系统正常工作时间和有关系统中处理器及内存的信息。

（8）/tmp。系统利用该目录存储暂存文件。不必计划在这里存储自己的暂存文件，程序将自动完成这一工作。

（9）/usr。/usr 目录包含系统中每个用户都使用的文件和程序。这里存放了随同 Linux 集成套件一起安装的大多数程序和实用工具，并且能够供普通账户（不仅仅是超级用户）使用。文件系统的层次结构规定了这个目录具有只读访问许可权，换句话说，用户不能改变 / usr 目录中的内容。

（10）/mnt。/mnt 存放临时的映射文件系统，我们常把软驱和光驱挂装在这里的 floppy 和 cdrom 子目录下。

（11）/sbin。/sbin 存放系统管理程序。

（12）/var。/var 包含系统产生的经常变化的文件，如打印机、邮件、新闻等假脱机目录、日志文件、格式化后的手册页以及一些应用程序的数据文件等。

（13）/root。/root 是超级用户的主目录，归系统管理员所有。

5.4 Linux 文件系统

从逻辑结构上看，Linux 系统中的文件是无结构的流式文件。从物理结构上看，Linux 采用混合多重索引结构，即将文件所占用盘块的盘块号直接或间接地存放在该文件索引节点的地址项中。在查找文件时只要找到该文件的索引节点就可以用直接或间接的寻址方式

获得指定文件的盘块号。这一节将介绍 Linux 文件的物理结构。

5.4.1　Linux 文件的物理结构

在 Linux 中根据文件的大小，形成小型、中型、大型和巨型等不同规模的文件，如图 5-14 所示为 Linux 的混合索引结构。

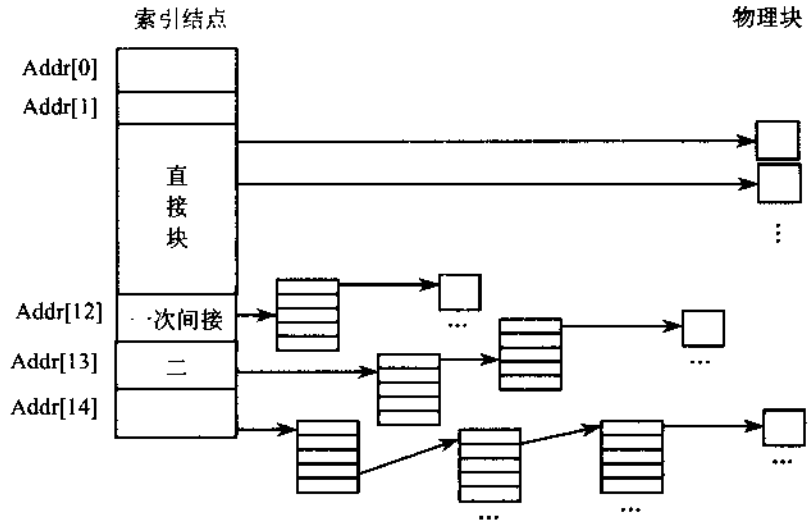


图 5-14　Linux 的混合索引文件结构

1. 寻址方式

（1）直接寻址方式。Linux 系统中的作业以中、小型为主，为了提高对文件的检索速度，宜采用直接寻址方式。在索引节点中建立 12 个地址项，用来直接存放该文件所在的盘块号，相应的盘块称为直接块。

（2）一次间接寻址方式。当文件较大时，Linux 系统提供了一次间接寻址方式。在这种寻址方式中，一次间接地址项中所对应的盘块（间接块）存放的不是文件所在的物理盘块号，而是直接块的块号表。为了通过间接块读取文件数据，核心是先读出间接块，找到相应的直接块项，然后从直接块中读取数据。

（3）多次间接寻址方式。对于大型和巨型的文件，Linux 系统又引入了二次间接寻址和三次间接寻址。二次间接项中存放的是一次间接块号表，三次间接项所对应的盘块中放有二次间接块号表。

2. 地址转换

Linux 系统利用地址转换过程将逻辑文件的字节偏移量转换为文件的物理块号，首先，将字节偏移量转换为文件逻辑块号及块内偏移量；然后，把逻辑块号转换为物理块号。

（1）将字节偏移量转换为文件的逻辑块号。在 Linux 系统中，被打开的文件在文件表项中有一个读、写指针，用以指示下次要读、写的字符。该偏移量是从文件第一个字符的位置开始计算的。每次读、写时，都要先把字节偏移量转换为文件逻辑块号和块内偏移量，

其转换方式为将字节偏移量除以盘块大小的字节数，其商为文件逻辑块号，余数为块内位移量。

（2）将文件逻辑块号转换为物理块号。根据逻辑块号可以求出对应的文件地址是直接地址还是间接地址，然后采用不同的方法进行转换。Linux 系统中直接地址项为 12 个。

1）直接地址。当文件逻辑盘块号小于 12 时，将逻辑块号转换为物理块号的方法是将文件的逻辑块号转换为索引节点的地址项下标，然后从该地址项中就可直接获得该文件的物理盘块号。

例如，设盘块大小为 1KB，某进程要访问字节偏移量为 7000 处的数据。首先将 7000 转换为文件逻辑块号 7000/1024=6，块内位移量 7000%1024=856。其逻辑块号小于 12，所以该块为直接块，从 addr[6]中读出对应的物理块号，该物理块第 856 字节处即为文件的第 7000 字节。

2）间接地址。当计算出的文件逻辑块号大于或等于 12 而小于 268 时，采用一次间接地址方式。将逻辑块号转换为物理块号的方法是从一次间接项中得到一级间接的盘块号，根据该间接块的内容计算一次间接块中的地址下标，即将文件的逻辑块号减 12，从相应下标的地址项中得到物理块号。

例如，盘块大小为 1KB，某进程要访问字节偏移量为 28000 处的数据。先将 28000 转换为文件逻辑块号 28000/1024=27，块内位移量为 28000%1024=352。由于逻辑块号大于 12 而小于 268，所以采用一次间接地址方式。先从一次间接项中得到一次间址的盘块号，再从一次间接块的地址下标为 15（27-12）的地址项中得到其物理块号，该物理块的第 352 字节即为要读取的数据。

当计算出的文件的逻辑块号大于或等于 268 时应采用多次间接地址寻址。

所以，在 Linux 系统中，采用混合索引结构的优点是对于小文件，访问速度快；对于大中型的文件，其文件系统也能很好地支持。

5.4.2 Linux 文件系统

与普通 UNIX 系统一样，Linux 使用树状的文件系统。每个系统都有一个根目录，从这里开始可以遍历整个系统中的所有文件，它没有驱动器和盘符的概念。

Linux 一个最重要特点就是支持许多不同的文件系统，如 ext、ext2xia、ext3、minix、umsdos、msdos、vfat、proc、smb、ncp、iso9660、sysv、hpfs、affs 和 ufs，这使 Linux 非常灵活，能够与许多其他操作系统共存。无疑随着时间的推移，Linux 支持的文件系统数还会增加。

1. 文件类型

Linux 把许多不同的东西看成文件。一般情况下，会碰到的文件类型有普通文件、目录文件、设备文件、链接。

其中，普通文件和目录文件的含义与 DOS 或 Windows 下的概念是类似的，普通文件就是一些数据的集合，而目录文件则是包含文件的容器。实际上，目录也是一种特殊的文件，只是它包含的内容是文件的存放信息。

设备文件被分成字符设备和块设备。字符设备通常是类似于终端那样可以用处理文本文件的方式来处理的东西，例如，Linux 在启动时打开了 6 个虚拟终端，分别称为 tty1～tty6，这些虚拟终端对应的文件是/dev/tty1～/dev/tty6。假设你拥有超级用户权限，就可以直接向这些 tty 上写文本，而这些文本则会显示在主机的控制台上。块设备代表了类似磁盘的东西，如磁盘、CD-ROM 或虚拟驱动器。一般情况下，每个块设备还要包含一些子设备，例如，第一个 IDE 硬盘被称为 hda，但是这个硬盘可以包含多个分区，那么，第一个分区称为 hda1，以下依次是 hda2、hda3、...等。

链接实际是一种文件别名，每个文件至少应该有一个名字，如果需要给文件增加一个名字以便访问，那么就需要建立一个链接。链接分为软链接和硬链接。硬链接指通过索引节点进行的连接。在 Linux 中，多个文件名指向同一索引节点是存在的。一般这种链接就是硬链接。硬链接的作用是允许一个文件拥有多个有效路径名，这样用户就可以建立硬链接到重要文件，以防止“误删”的功能。因此，由于对应该目录的索引节点可以有一个以上的链接，只删除一个链接并不影响索引节点本身和其他的链接，只有当最后一个链接被删除后，文件的数据块及目录的链接才会被释放，也就是说，文件才会被真正删除。硬链接与文件原有的名字是平权的，在 Linux 看来它们是等价的。由于这个原因，硬链接不能连接两个不同文件系统上的文件。软链接也叫符号链接（Symbilc Link），实际上只是一段文字，里面包含着它所指向的文件的名字，系统看到软链接后自动跳到对应的文件位置处进行处理。软链接与硬链接的主要区别是，软链接和它指向的文件不是处于同等地位。例如，为文件 foo 建立一个叫 fool 的符号链接，对 fool 的访问具有与直接访问 foo 相同的效果。如果删除了 fool，foo 仍然存在；但是如果删除了 foo，fool 虽然还存在，但它指向一个空的链接，相当于 Windows 下的快捷方式。

2. 使用文件系统

通常，对于服务器，安装的同时就为它设计好了文件系统，特别是如何分区，某个分区应该连接到哪个目录下等。我们也需要对文件系统进行维护，例如为服务器增加一个硬盘，或者解决文件系统混乱的灾难性问题等。另外，诸如 CD-ROM、软磁盘，或者服务器上的共享目录等需要经常更换，在 Linux 中，用文件系统的安装和拆卸来解决这样的问题。

一个文件系统可以连接到已有的 Linux 系统中，也可以把连接上的文件系统从整个目录树中脱离，这是用 mount 和 umount 命令实现的。

（1）mount 命令。功能是安装文件系统，格式为：

mount 开关　系统设备文件名　　文件系统挂接点

开关是一些命令参数，例如，假定需要使用某个软盘上的文件，将它插入驱动器 A，执行 mount -t /dev/fd0　/mnt/floppy，然后软盘的文件系统就被挂到/mnt/floppy 目录下，相当于 dos 的 a:\ff 的文件现在的名字是/mnt/floppy/ff。注意，mount 命令要求在挂接之前连节点（在这个例子中是/aa/floppy）必须存在，否则无法执行。

一般总是把文件系统连接到某个空目录。假如/mnt/floppy 目录在连接之前就包含某些文件，那么执行上述命令的结果将使/mnt/floppy 目录的内容变为软磁盘上的内容，而原有的内容在卸掉这个文件系统之前将不可访问。

/dev/fd0 是与驱动器 A 对应的设备，相应地驱动器 B 是/dev/fd1。硬盘的情况比较复杂，因为 Linux 是利用设备接口来描述设备的，例如，第一个 IDE 接口上的主盘是/dev/hda，从盘是/dev/hdb，第二个 IDE 接口上的两个盘是/dev/hdc 和/dev/hdd。由于硬盘使用的是分区，所以必须使用子设备名，例如，把/dev/hda 上的第一个分区连接到/aa/diskc 上的命令是 mount /dev/hda1 /mnt/diskc。相应地，第二个分区是/dev/hda2，以此类推。Linux 对 IDE 光驱用硬盘的命名方式，不过光盘不存在分区问题，所以可以直接使用设备名，例如把第二个 IDE 从盘上的 CD-ROM 连接到/mnt/cdrom 的命令是 mount /dev/hdd /mnt/cdrom。连接了一个文件系统之后，就可以对这个系统对应的目录进行操作，效果就是操作对应的文件系统。

（2）umount 命令。该命令的功能是卸下文件系统，格式为：

umount 连接点

例如，在上面的例子中，把连接在/mnt/floppy 的软盘文件系统卸下的命令是 umount /mnt/floppy / 。

注意，由于这个过程会改变目录结构，因此必须在没有任何进程访问/mnt/floppy 下的文件，也没有任何用户的当前目录是/mnt/floppy 或它的下级目录时才能完成，否则会返回一个“device is busy”的错误。

mount 和 umount 都需要超级用户权限才能执行，它们都支持-a 选项，mount –a 挂接在/etc/fstab 中定义的所有可挂接文件系统；umount –a 卸下所有定义在/etc/mtab 中的已挂接文件系统。每当挂接一个文件系统时，系统自动在/etc/mtab 中增加一条记录，当这个系统被卸下时自动把记录清除。直接输入不带参数的 mount 指令将会显示当前已挂接的文件系统。

3. 虚拟文件系统 VFS

Virtual File System（或称为 Virtual Filesystem Switch）是 Linux 内核中的一个软件层，用于给用户空间的程序提供文件系统接口，它也提供了内核中的一个抽象功能，允许不同的文件系统共存。

实现 Linux 虚拟文件系统要使得它对文件的访问尽可能快、尽可能高效，而且一定要确保文件和数据的正确性，这两个要求彼此是不对称的。Linux VFS 在安装和使用每一个文件系统时都在内存中高速缓存信息。在文件和目录创建、写和删除时，这些高速缓存的数据被改动，必须非常小心才能正确地更新文件系统。如果能看到运行的核心中的文件系统的数据结构，就能看到文件系统读写数据块，描述正在访问的文件和目录的数据结构会被创建和破坏，同时设备驱动程序会不停地运转，获取和保存数据。这些高速缓存中最重要的是 Buffer Cache，它在文件系统访问底层的块设备时结合进来。当访问块时它们被放到 Buffer Cache，根据状态的不同放在不同的队列中。Buffer Cache 不仅缓存数据缓冲区，也帮助管理块设备驱动程序的异步接口。

4. EXT2

EXT2 是 Linux 用户最常用的文件系统。各种 Linux 的系统发布都将 EXT2 作为操作系统的基础。EXT2 文件系统中的数据是以数据块的方式存储在文件中的，这些数据块的大小相同，在 EXT2 创建时设定。EXT2 用一个 inode 数据结构描述系统中的每一个文件，定义了系统的拓扑结构。一个 inode 描述了一个文件中的数据占用了哪些块以及文件的访问权限、

文件的修改时间和文件的类型。EXT2 文件系统中的每一个文件都用一个 inode 描述，而每一个 inode 都用一个独一无二的数字标识。文件系统的 inode 都一起放在 inode 索引表中。EXT2 的目录是简单的特殊文件（它们也使用 inode 描述），包括目录条目的 inode 指针。

根据 inode 所存放的位置，i 节点又可分为磁盘 i 节点和内存 i 节点。磁盘 i 节点静态存放在外部存储器，包含了整个文件的完整说明信息和物理块分布情况，内存 i 节点是为了减少设备存取次数、提高文件访问效率而在内存特定区域中建立的磁盘 i 节点的映像。

（1）磁盘 i 节点结构。每个磁盘 i 节点占 128 个字节，包括文件名称、类型、文件长度、所占用的物理块数、创建和最近一次访问时间等基本信息，以及链接数、拥有者用户名、用户组名以及存取控制表等与文件共享、保护和保密相关的信息。代表磁盘 i 节点数据结构是 ext2_inode，它被定义在 include/linux/ext2_fs_i.h 文件中。其结构为：

```
Struct ext2_node
 {
_u16   i_mode;                          /*文件类型及访问权限*/
_u16   i_uid;                           /*文件主标识符*/
_u_32  i_size;                          /*以字节为单位的文件大小*/
_u_32  i_atime                          /*最后一次访问文件时间*/
_u 32  i_ctime;                         /*索引节点最后改变的时间*/
_u_32  i_mtime;                         /*文件内容最近修改时间*/
_u_32  i_dtime;                         /*文件删除时间*/
_u16   i_links_count;                   /*文件连接计数*/
_u32   i_blocks;                        /*文件的数据块块数*/
_u16   i_gid;                           /*用户组标识符*/
_u32   i_flags;                         /*文件标志*/
union  osd1                             /*特定的操作系统信息*/
_u32 [EXT2_N_BLOCKS]  i_block;          /*指向数据块的指针*/
_u32   i_version                        /*文件版本*/
_u32   i_file_acl;                      /*文件访问控制表*/
_u32   i_dir_acl;                       /*目录访问控制表*/
_u32   i_faddr                          /*片的地址*/
union  osd2                             /*特定的操作系统信息*/
}
```

每个 i 节点的索引表的前 12 项记录文件是直接物理块的位置，后面 3 项分别是间接、二级间接和三级间接物理块索引，如图 5-14 所示，这样的方式既可以保证小文件能够快速存取，又可以适应大文件存储扩展的需要。

（2）内存 i 节点。除了包含磁盘 i 节点的说明信息之外，还增加了当前文件打开状态信息，代表内存 i 节点的数据结构是 ext2_inode_info，在 include/linux/ext2_fs_i.h 文件中。

```
Struct ext2_inode_info{
  _u32  i_data[20];                     /*数据块指针数组*/
  _u32  i_flags;                        /*管理标志位*/
  _u32  i_faddr;                        /*文件地址*/
_u8  i_frag_no;                         /*第一个片号*/
_u8  i_frag_size;                       /*片的尺寸*/
```

```
_u16  i_osync;                    /*同步信息*/
_u32  i_file_acl;                 /*文件访问控制链表*/
_u32  i_dir_acl;                  /*目录访问控制链表*/
_u32  i_dtime;                    /*文件删除时间*/
_u32  nit_used_l;                 /*FIX:not used*/
_u32  i_block_group               /*inode 所在的块*/
_u32  i_next_alloc_block;         /*下一个要分配的块*/
_u32  i_next_alloc_goal;          /*下一个要分配的目标对象*/
_u32  i_prealloc_block;           /*要分配的第一个数据块*/
_u32  i_prealloc_count;           /*要分配的数据块总数*/
_u32  i_high_size;
int i_new_inode_l;
}
```

（3）EXT2 的目录项数据结构。EXT2 采用动态方式管理它的目录，用一个单项链表示目录项，每个目录项的数据结构如下：

```
#define EXT2_NAME_LEN  255
 Struct ext2_dir_entry
 {
_u32 inode;                       /*索引节点号*/
_u16 inode;                       /*目录项长度*/
_u16 inode                        /*文件名长度*/
charname [EXT2_NAME_LEN] ;        /*文件名*/
}
```

目录项的长度是可变的，因为它的最后一个名字成员是可变长数组，而且文件名是以字节的整数倍存放，并在必要时用 null 字符（\0）填充文件名的末尾。

当要删除一个目录时，要将目录项中的 i 节点置为 0，并把目录项从链表中删除，目录项所占用的空间被合并到前一个目录项空间中。

另外，Linux 中有一种专门用于交换分区的 swap 文件系统，Linux 使用整个分区作为交换空间，而不像 Windows 那样使用交换文件。一般这个 swap 格式的交换分区是内存的 2 倍。

5.5 文件存储空间的管理

文件存储空间的管理是对空闲空间的管理，建立关于空闲盘块的数据结构，给文件分配存储空间提供依据。不同的外存空间分配方式（文件物理结构）可能采用不同的空闲盘块的数据结构。

下面介绍几种常见的存储管理方式。

5.5.1 空闲空间表法

系统为外存上的所有空闲区建立一张空闲表，每个空闲区对应一个空闲表项，包括序号、空闲区的第一个盘块号、该区的空闲盘块个数以及对应的空闲物理块号等，如图

5-15 所示。

序　号	第一个空闲块号	空闲块个数	物理块号
1	2	4	(2,3,4,5)
2	19	3	(19,20,21)
3	17	2	(17,18)
……	……	……	……

图 5-15　空闲空间表

这种管理方法适用于采用顺序结构的文件。空闲盘区的分配与内存的动态分配类似，同样是采用首次适应算法、循环首次适应算法等。例如，系统为某新创建的文件分配空闲盘块时，先顺序地检索空闲表的各表项，直至找到第一个大小能满足要求的空闲区，再将该盘区分配给用户（进程），同时修改空闲表。系统在对用户所释放的存储空间进行回收时，也采取类似于内存回收的方法，即要考虑回收区是否与空闲表中插入点的前区和后区相邻接，若相邻接，则把它们合并成一个大的空闲区，记在一个表项中。

5.5.2　空闲块链法

这种方法是将磁盘上的所有空闲存储空间，以盘块为单位拉成一条链，用一个指针指向第一个空闲块，而各个空闲块中都含有下一个空闲区的块号，最后一块的指针项记为 NULL，如图 5-16 所示。

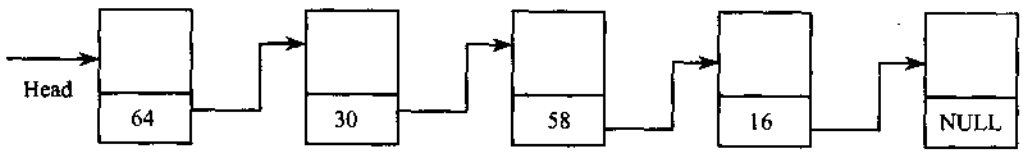


图 5-16　空闲块链

当用户因创建文件而请求分配存储空间时，就从链表头依次取下适当数目的空闲块，分配给用户。当删除文件时，就把新释放的块从链表插入，并使头指针指向最后释放的那一块。

这种管理方法适用于非连续分配。由于各个空闲块的链接指针隐含在空闲磁盘块中，因此管理时所需的额外开销很少，但工作效率较低，因为在空闲块链上增加或删除空闲块时需要做许多 I/O 操作。

5.5.3　位示图法

1. 位示图

由于磁盘被分块后，每一块的大小相等，块数固定，所以可采用一种存储空间的管理方法——位示图法。这种方法是利用二进制的一位来表示磁盘中每一个盘块的使用情况，

磁盘上的所有盘块都有一个二进制位与之对应，从而由所有盘块所对应的位构成一个集合，即位示图。采用位示图的具体做法是，为所要管理的磁盘设置一张位示图。位示图的大小由磁盘的总块数决定，每一个盘块与位示图的一个二进制位对应。如果盘块是空闲的，对应位为 0；如果盘块已分配出去，对应位为 1。

2. 空闲盘块的分配

当分配存储空间时，查找位示图中对应位为 0 的位，然后将其转换为对应的物理块号，将其分配给申请者，并将该位置 1。根据位示图为某文件分配盘块的具体过程如下：

（1）顺序扫描位示图，找出其值为空闲即 0 的二进制位。

（2）将二进制位的行/列号转换为与之对应的盘块号，假设找到的值为 0 的二进制位位于位示图的第 i 行、第 j 列，则相应的盘块号为 b=n*(i-1)+j，其中 n 代表每行的位数。

（3）把盘块号给该文件，同时修改位示图中的二进制位 Map[i,j]=1。

3. 盘块的回收

删除文件时，需要回收存储空间，即把原来所占用的物理块归还给系统，并将物理块对应位置 0。根据位示图回收盘块的具体过程如下：

（1）将回收盘块的盘块号 b 转换成位于位示图中的行号 i 和列号 j:i=(b-1)/n+1；j=(b-1)%n+1。

（2）修改位示图中的二进制位 map[i,j]=0。

位示图的优点是占用的存储空间少，因此可以将位示图全部装入内存，使得盘区的分配和回收都可以高速地进行。但分配时，需要顺序扫描位示图，且物理块号在图中并未直接反映出来，需要计算所在的物理块号。

5.5.4 空闲块成组链接法

空闲空间表法和空闲块链法都不适合用在大型的文件系统中，因为会使空闲表或空闲链太长。在 Linux 系统中，采用了一种改进的方法，称为“成组链接”法。这种方法兼备了上述两种方法的优点而克服了两种方法均有的表太长的缺点。

1. 空闲盘块的组织

（1）将文件区中的所有空闲盘块依次进行分组，如将 100 个空闲盘块划分为一组，将组中的第一块称为“组长块”，第一组为 99 块。从第二组开始，每块都为 100 块，剩下的块归并为最后一组。

（2）将每一组的盘块总数和该组所有盘块号放入后一组的“组长块”中，这样，由每一组的第一个盘块构成一条链。

（3）第一组中只有 99 个空闲盘块，为了管理的需要，把它的总块数仍记为 100，而第二组的“组长块”中标记的第一组的首块号为 0，作为空闲盘块的结束标志。

（4）将最后一组的盘块总数和空闲盘块号存入空闲盘块专用栈中，这样，空闲盘块的分配和回收就在这个专用栈中进行，如图 5-17 所示。

2. 空闲块的分配

当需要分配空闲盘块时，先把专用栈中表示栈深（即栈中有效元素的个数）的数值减

1，这里是 100-1=99，以 89 作为检索专用块中空闲盘块号栈的索引。如图 5-17 所示，盘块号为 151，它就是当前分配的第一个空闲盘块。如果需要分配多块，重复上述操作。

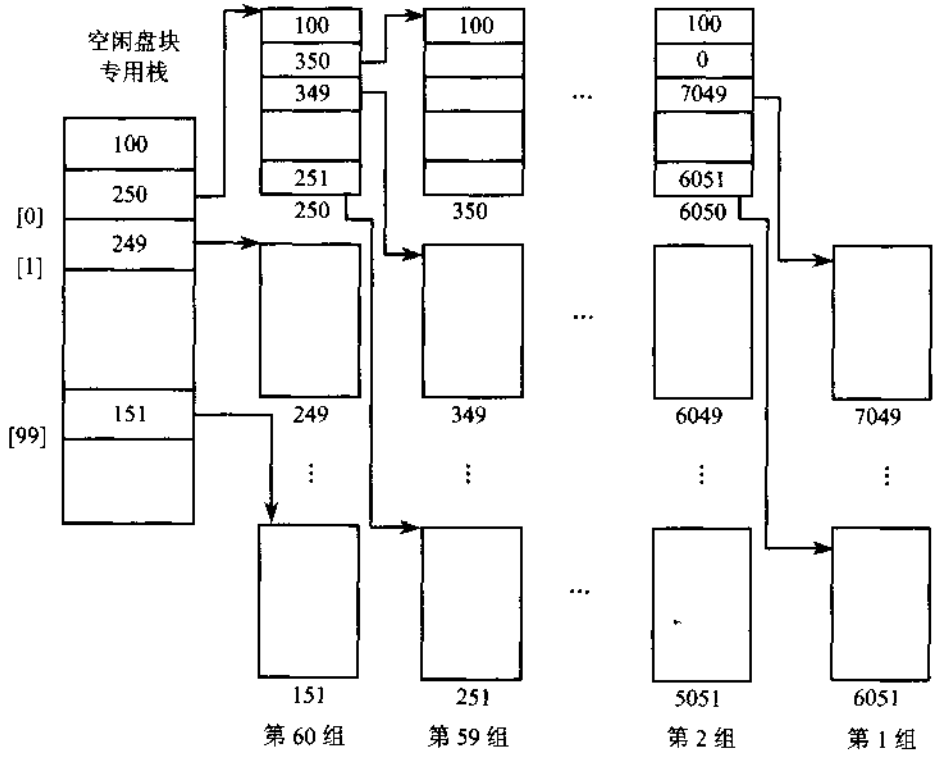


图 5-17　空闲块成组链接法

需要注意的是，当栈深值减 1 后，其值等于零，此时系统需要做特殊处理，因为这一组的第 1 个盘块（总是在一组中的最后被分配出去的盘块）包含它前一组空闲块的信息。因此，在把这一块分配出去之前，应先把它记录的信息复制到专用栈中，然后再分配出去。分配时还要注意一个问题，若要把第二组的第 1 块分配出去，那么先把它所含信息复制到专用栈中，这就意味着只有 99 个盘块可供分配了。继续分配，直到把 99 个盘块全部分配出去。若再申请空闲盘块，栈深值减 1 变为 0，此时发现对应的空闲盘块号为 0，表示所有的空闲盘块号都已经分配出去了，对应进程只能等待。

3. 空闲块的回收

当删除文件，需要回收盘块时，将回收盘块的盘块号记入空闲盘块号栈的顶部，并将空闲盘块数加 1。当栈中的空闲盘块号数目已达到 100 时，表示栈已满，若还要回收盘块，则需进行特殊处理：将栈深值和各空闲块的块号写到要释放的新盘块中；将栈深值及栈中盘块号清 0；将新回收的盘块写入相应单元中，栈深值加 1。新回收的块将作为新组的组长块。

5.6 文件共享与安全性

5.6.1 文件的共享

文件共享是指多个用户或进程可以共同使用一个或多个文件。利用文件共享可以减少大量的外存空间和内存空间，同时为用户完成各自的任务带来很大方便。随着计算机技术的发展，文件共享已不限于单机系统，而扩展到了全球的计算机网络系统。

实现文件共享的方法有两种：绕弯路法和连接法。

1. 绕弯路法

这是早期的 MULTICS 等操作系统中所采用的一种共享方法。在该方法中，允许每个用户获得一个“当前目录”，当所访问的文件不在当前目录时，可以通过“向上走”的方式去访问其上级目录，一般用“*”表示目录的父目录。

2. 连接法

在树形结构目录中，当有两个（或多个）用户要共享一个子目录或文件时，必须将共享文件或子目录链接到两个（或多个）用户的目录中，以便能方便地找到该文件，此时该文件系统的目录结构已是非循环树形目录结构（Directed Acyclic Graph），如图 5-18 所示。

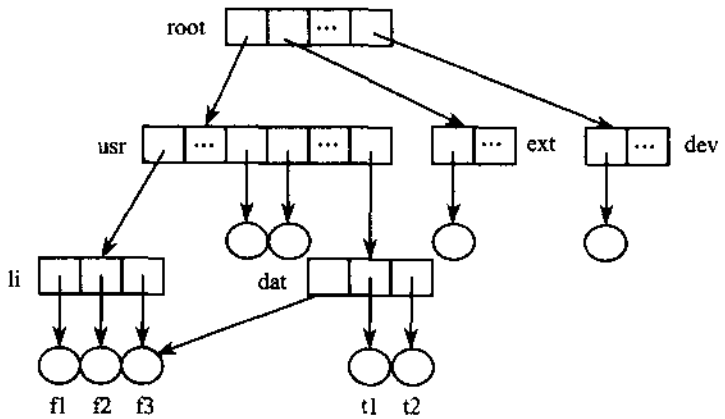


图 5-18 文件的连接

建立连接的基本思想是用一个文件目录项直接指向要共享文件的目录项，从而在两个文件之间建立起一种等价的关系。具体做法是在文件主允许的情况下，一个用户在他的文件目录中开辟一个目录项，该目录项直接指向所要共享的那个文件的目录项。

为了建立目录与共享文件之间的链接，并顺利实现共享，可以引用索引节点，即诸如文件的物理地址及其他的文件属性等信息，不再放在目录项中，而放在索引节点中。在文件目录中只设置文件名及指向相应索引节点的指针。

例如，用户 dat 要共享文件 li 中名为 f3 的文件。用户 dat 在自己的目录中开辟一个目录项，其指针直接指向用户 li 的文件 f3。这样，在用户 li 和用户 dat 之间就建立了一个连

接关系，用户 dat 可以直接访问 f3 这个文件，如图 5-19 所示。

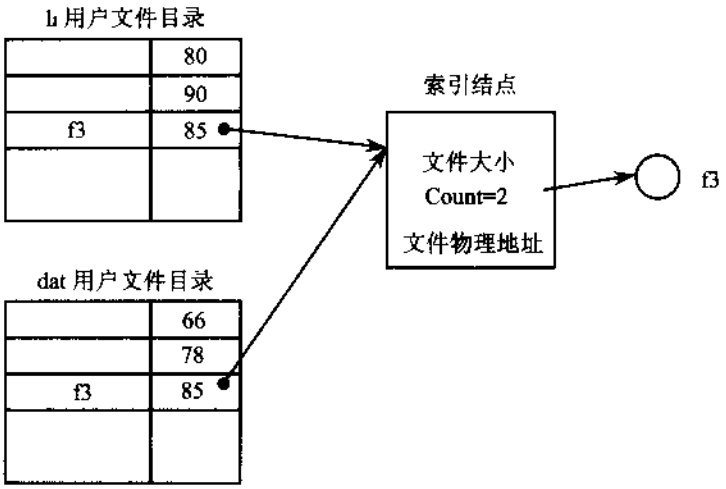


图 5-19　基于索引结点的共享方式

在 Linux 系统中，对文件的连接命令是 ln，命令格式为：

ln　源目标文件名　新链接文件名

例如，要将当前目录（/usr/li）下的文件 f3 连接到目录/usr/dat 中，可使用命令：

ln　f3　/usr/dat

连接后结果如图 5-18 所示。若当前目录是 dat，可使用命令 cat f3 来显示文件内容。

连接的具体实现过程是：

（1）根据源文件名检索目录树，找到对应的文件控制块，并复制到内存中。

（2）根据新的文件名检索目录树，若找到该文件，则判断出错。

（3）若未找到，则在新目录中登记该文件的目录项，并增加源文件的连接计数。

值得注意的是，连接文件并不是创建文件，只是增加一条共享文件的路径。

5.6.2　文件的安全性

文件的共享给人们带来很多好处和方便的同时，也潜在着诸多不安全的因素，如人们有意或无意的行为使文件系统中的信息遭到破坏或丢失，自然灾害造成的机器损坏，系统故障造成的数据破坏或丢失等，这些给系统的安全性带来很大的影响。因此，在现代计算机系统中，不仅为用户提供共享的便利，而且要充分注意到系统和数据的安全性和保密性。

文件保护是指防止未经授权的用户使用文件或文件主的错误操作对文件造成破坏。通常可以采用口令、密码、存取控制矩阵、存取控制表等保护机制，来达到保护文件不受侵犯的目的。

1. 口令方式

用户在文件创建时，可以为每一个文件设置口令（Password），并被记录在文件说明

信息中。文件系统在用户试图访问该文件时，首先要求用户提供口令并与文件说明信息中记录的口令进行比较，若匹配用户才能够存取该文件。

通过这样的方法，可以简单地实现文件的共享与保密。口令验证的过程比较简单，占用空间少；保密性能相对较差，易被窃取。

2. 密码方式

为了防止破坏或泄密，对一些重要信息可采用密码方式来存储。密码（Encryption）方式在文件创建时，文件主在文件写入存储设备之前，通过特定的算法和加密密钥对文件内容进行编码加密，读取文件时，必须提供相应的解密密钥进行译码解密。只有确切知道解密密钥的用户才能够读出被加密的文件。

密码方式保密性强，节省存储空间，但是加密和解密要花费很多时间，增加系统开销。

口令方式中有时也采用一定的加密技术。加密口令方式只是对口令本身加密，加密后的口令仍然存放在文件说明信息中，而密码方式是对整个文件进行加密，加密时采用的密钥掌握在用户自己手中，因此，密码方式的保密性强，但是要耗费大量的处理时间。以上两种文件存取权限控制方法是针对特定文件的，在多用户系统中，还可以针对特定的用户进行文件存取权限控制。大多数系统，包括 Linux，用户身份验证都采用用户口令方式，建立用户时，设置其口令，口令以加密的形式存放在系统中，每次用户登录都要提供口令，并与系统中存放的数据进行比较，如果相同，用户登录成功，否则，拒绝用户登录。在这样的系统中，多个用户可以同时使用计算机，每个用户都拥有自己的文件，这些文件的存取使用控制权由用户拥有。

3. 存取控制矩阵

存取控制矩阵是指在整个系统中建立一个二维表，一维列出系统中的所有用户名，一维列出系统中的所有文件名，矩阵的每一个元素都表示该元素对应的用户对文件的存取控制权限，如图 5-20 所示。当用户对某个文件进行操作时，系统查找矩阵中该用户和该文件所对应的元素的值，只有当该元素表明有访问许可时，才能进行具体的访问。

文件名 / 用户 / 权限	AA	AC	LI	FLAG	POIN
用户 1	RW	R	RE	RWE	
用户 2	E		R		RW
用户 3	RWE	E		E	
…					

图 5-20 存取控制矩阵

存取控制矩阵可以对系统中所有文件的存取权限进行完整有效地控制。但是当系统中用户和文件数很大时，存储和管理这样的矩阵需要比较大的系统开销，并且用户每访问一个文件，都要对矩阵的一行进行搜索，访问效率比较低。

4. 存取控制表

存取控制表是把文件主和相关的用户对某个文件的访问按照某种关系分类，不同的用户类赋予不同的文件存取权限。Linux 系统中，文件的存取权限分别赋予文件主、文件主所在用户组和其他用户组这 3 类用户，每一类访问的权限设置 3 位，分别为读、写和执行，这样就相当于每个文件只需要记录 9 位数据就可以进行权限控制了，如图 5-21 所示。

实际上，每个文件的存取控制表都相当于整个存取控制矩阵中的一列元素。存取控制矩阵中的每一列对应于一个文件，去掉空元素，然后把所有用户按照文件所有者、所有者所在组和其他用户组简化后，就可以得到该文件的存取控制表，如图 5-21 所示。

用户 \ 权限 \ 文件名	文件
文件主	RWE
同组用户	RE
其他用户	E

图 5-21　存取控制表

在 Linux 系统中，每一个文件都在文件说明信息中保存着自己的文件存取控制表，只有符合存取控制表中规定的条件，才允许进行具体的访问。

需要说明的是，设置文件访问口令和文件加密属于文件级的访问权限控制。在多用户系统中，必须进行用户验证，文件系统针对特定的用户，可以通过存取控制矩阵或存取控制表来设置对文件的存取权限，这类方法可以认为是基于用户身份的存取权限控制，属于用户级的访问权限控制。

5.7　Linux 系统的文件操作

在 UNIX、Linux 等操作系统中，把包括硬件设备在内的能够进行字符流式操作的内容都定义为文件。Linux 系统中文件的类型包括普通文件、目录文件、连接文件、管道（FIFO）文件、设备文件（块设备、字符设备）和套接字。操作系统根据文件的类型来处理文件。

下面介绍一下 Linux 系统中常用的命令。

5.7.1　文件操作命令

1. 文件显示命令

（1）cat 命令。

功能：用于连接并显示指定的一个或多个文件的有关信息，它的使用权限是所有用户。

格式：cat[-选项]文件名

例 1：cat 命令一个最简单的用处是显示文本文件的内容。

$ cat mm.c

在屏幕上显示 mm.c 的内容。

例 2：有时需要将几个文件处理成一个文件，并将处理结果保存到一个单独的输出文件中。cat 命令在其输入上接收一个或多个文件，并将它们作为一个单独的文件打印到它的输出中。例如，把 f1 和 f2 的文件内容加上行号（空白行不加）后，将内容附加到一个新

文本文件 f3 中：

$ cat f1 f2 f3

需要说明的是当文件内容过多时，就带来一个问题，因为文本在屏幕上迅速地闪过，用户来不及看清其内容，因此，当文件内容较大时，一般可用 more 等命令分屏显示，以免因屏幕滚动太快而无法看清。

（2）more 命令。

功能：在终端屏幕按屏显示文本文件。

格式：more [-选项]　文件名

使用 more 命令时，显示一屏后暂停，可看到屏幕的左下方有一个“--more--”的信息，系统还将同时显示出已显示文本占全部文本的百分比。这时若按下回车键，则会显示下一行；若按下空格键，则会显示下一屏。more 命令很适宜阅读大型文件。

（3）less 命令。

功能：与 more 命令相似，逐屏显示文件，不同之处在于 less 命令显示文件时可以前翻或后翻。

格式：less　文件名

使用 less 命令时，按下空格键，则会显示下一屏，按下回车键则一行行地下翻，按下 b 键往上翻一页。

（4）head 命令。

如果用户希望查看一个文件究竟保存的是什么内容，可以只查看文件的前几行，而不必浏览整个文件。用 head 命令只显示文件或标准输入的前几行。

格式：head [n] 文件名

功能：显示指定文件的前若干行。

说明：该命令显示每个指定文件的前 n 行。如果没有给出 n 值，默认设置为 10。如果没有指定文件，head 就从标准输入读取。例如，显示文件 example.c 的前 3 行：

$ head -3 example.c

（5）tail 命令。

如果用户想查看文件的尾部，可以使用 tail 命令。

格式：tail [+/-num] [参数] 文件

说明：该命令显示一个文件的指定内容，把指定文件的指定显示范围内的内容显示在标准输出上。如果没有给定文件名，则使用标准输入文件。

tail 命令中各选项的含义为：

+num：从第 num 行以后开始显示。

num：从距文件尾 num 行处开始显示，如果省略 num 参数，系统默认值为 10。

l：以文本行为 num 的计数单位。与参数选项+num 或-num 选项同时使用时，num 表示要显示的文本行行数。

c：以字节为 num 的计数单位。与参数选项+num 或-num 选项同时使用时，num 表示要显示的字符数。

1. c 选项可以省略，系统默认值为 l，即按行计数，例如：

$ tail -4 example.c

将显示文件 example.c 的最后 4 行。

2. ls 命令

功能：用于显示目录内容，类似 DOS 下的 dir 命令，它的使用权限是所有用户。

格式：ls [选项] [参数]

“[]”中的内容可有可无，也可以有多个。选项一般为英文字母，字母前有一个连字符“-”。ls 命令相当于 DOS 中的 dir 命令。

例如 ls –l，选项-l 以长格式显示，每行列出的信息依次为文件类型、存取权限、连接数、文件主、组号、大小（字节）、日期、文件名或目录名。例如：

-rwxr--r--1 root　news 12345　dec 29 2004 profile1

-rw-r--r--1 root　news 43217　dec 29 2003 services

注意：第一列中包含 10 个字符，其中第一个字符用于确定文件的类型。

- 普通文件

d 目录文件

c 字符文件

b 块文件

后面 9 个字符代表文件的许可权限，3 个字符一组。第一组是文件主的存取权限，第二组是同组用户的存取权限，第三组是其他用户的存取权限。其中，r 是读权限，w 是写权限，x 是执行权限。

3. cp 命令

功能：将给出的文件或目录复制到另一文件或目录中，如同 DOS 下的 copy 命令一样，功能非常强大。

格式：cp　源文件　目的地

例如，$cp m1.c m2.c 将把 m1.c 的内容复制到 m2.c 中。$cp m1.c m2.c mm 把文件 m1.c 和 m2.c 复制到目录 mm 中。

cp 命令与 DOS 的 copy 命令大致相同。需要注意的是，使用 cp 时一定要有目的地才行，在 DOS 中做 copy 时，有时可省略目的地，但在 Linux 中就无法这样做。在 cp 中可以使用通配符，如“*”、“？”等，例如，要将 usr 目录内的所有文件，但不包括隐藏文件，复制到根目录下的 dev 内，其命令为：

$cp　/usr/*　/dev

若要复制所有小写字母开头的隐藏文件，其命令为：

$cp　/usr/.[a-z]*　/dev

4. mv 命令

功能：用来为文件或目录改名，或者将文件由一个目录移入另一个目录中，它的使用权限是所有用户。该命令如同 DOS 命令中 ren 和 move 的组合。

格式：mv 文件名称 搬移的目的地（或更改的新名）

例如，命令行$mv f1.c ff.c 将把 f1.c 移动到 ff.c。如果 ff.c 不存在，那么执行后，相当于把 f1.c 重新命名为 ff.c；如果 ff.c 已存在，则 ff.c 原来的内容被覆盖。

文件更名与移动文件的操作原理是一样的，差别只是路径不同。mv 命令通常被用来移动文件，例如，把现在所在目录中的 netscape 文件移到/usr 内，可用命令行$mv netscape /usr 实现。

5. rm 命令

功能：删除文件。

格式：rm 文件

例如，$rm *.c 将删除当前目录下后缀名为.c 的文件。

rm 与 DOS 的 del 命令有些类似，但 rm 的功能和杀伤力都相对大得多。只要是文件，不管是否隐藏，或是文件使用权限是否设置成只读，rm 皆可删除，在此要注意的是已删除的文件是无法恢复的，所以使用 rm 时要特别小心。例如，删除 ta 目录下的 file 文件，可用 $ cd ta;rm file。在 Linux 中要一次键入两个不同的命令时，只需在命令与命令之间加上分号，Linux 便会依照排列的先后次序执行，在本例中，先执行 cd ta，再执行 rm file。

6. find 命令

功能：在目录中搜索文件，使用权限是所有用户。

格式：find [路径][选项][表达式]

如果不写路径，那么默认为当前目录。

例 3：根据文件名查找。

我们想要查找一个文件名是 l.conf 的文件，可以使用如下命令：

find / -name l.conf

find 命令后的“/”表示搜索整个硬盘。

例 4：快速查找文件。

根据文件名查找文件会遇到一个实际问题，就是要花费相当长的时间，特别是大型 Linux 文件系统和大容量硬盘文件放在很深的子目录中时。如果知道了这个文件存放在某个目录中，那么只要在这个目录中往下寻找就能节省很多时间。例如 smb.conf 文件，从它的文件后缀.conf 可以判断这是一个配置文件，应该在/etc 目录内，此时可以使用命令 find /etc -name smb.conf。

5.7.2 目录操作命令

在 Linux 系统中，文件采用的是树形目录层次结构，目录层次的顶部是根目录，用符号“/”表示。常用的目录命令如下所示。

1. pwd 命令

功能：显示当前工作目录的全路径名。

格式：pwd

2. cd 命令

功能：改变当前工作目录。

格式：cd 目录名

cd 另一个功能是不管在哪个目录内，只要输入 cd 命令，不用接任何参数，就可回到用户目录（Home Directory）内。

3. 目录的创建、删除及更名

（1）mkdir/rmdir 命令。

功能：创建目录和删除目录。

格式：mkdir　目录名

例如，mkdir　rr 在当前目录下创建子目录 rr。rmdir md 将当前目录下的子目录 md 删除。

操作方法与 DOS 中的 md、rd 完全相同，差别只是命令的名称不同而已。另外，使用 rmdir 时，要确保该目录内已无任何文件存在，否则该命令不成功。

（2）rm 命令。

功能：删除目录。

格式：rm -r 目录

选择项“-r”表示删除目录的同时一并删除目录内的子目录及文件，这个功能相当于 DOS 中的 deltree 功能，通常使用 deltree 时会有提示信息，但在 Linux 中使用 rm 没有任何提示信息，值得注意的是已删除的文件是无法挽救回来的，所以使用 rm 时要特别小心。当某个目录不再需要，可以用命令$rm -r /md 删除。

（3）.mv 命令。

功能：目录更名或搬移。

格式：mv 目录名　目的地（更改的新名）

具体操作与文件更名一样，例如，要把现在所在目录中的 usr1 目录移到/bin 内，可用 $mv　usr1　/bin 命令行。

4. cp 命令

功能：带目录复制。

格式：cp -r 目录　目的目录

若要复制一个目录，但该目录内还有好多子目录时，可以使用选择项-r，来复制目录内的子目录及文件，并且复制时会自动建立目录，此功能相当于 DOS 内的 xcopy。例如，要将 usr 目录内的所有文件（包括目录，但不包括隐藏文件）复制至根目录下的 ext 内，使用命令行#cp -r /usr/*　/ext。

5.7.3　存取控制命令

Linux 是一个多用户系统，系统中的个文件或目录可能被多个用户同时访问，以实现资源共享。为保护用户自己的数据和信息防止他人非法使用，系统采用了一套存取权限的机制来限制对文件的访问。各个拥有者（owner）、组成员（group）和其他用户有不同的使用权限。用户可用下面的命令规定自己主目录下的文件权限。

1. chown 命令

功能：改变文件拥有者。

格式：chown　用户帐号　文件或目录名

若（假设帐号是 x1）有一个名为 file1.c 的文件，其拥有权要给予另一位账号为 x2 的同事，则可用 chown 来完成此功能，当改变完文件拥有者之后，该文件虽然在 x1 的 home 目录下，但该用户已经没有任何修改或删除这个文件的权限了。例如：

-rw------1　　x1 users　6 Oct 4 12：50　file1.c

#chown x2 file1.c

-rw------1　　x2 users　6 Oct 4 12：50　file1.c

需要说明的是，只有该文件或目录的拥有者和 root 用户才可以使用该命令。

2. chgrp 命令

功能：改变文件的所属组。

格式：chgrp　组名　文件或目录名

该命令和 chown 用法一样，其功能是把文件或目录所属组改成另一个组。

例如：

-rwxr-x---　1 root tem　1100 Oct 12 06：48　sun

chgrp sys /sbin/sun

执行 chgrp 把 sun 所属组改成 system 组

-rwxr-x---　1 root sys 1100 Oct 18 06：52 sun

需要说明的是，只有 root 用户才可使用该命令。

3. chmod 命令

功能：修改文件或目录的存取权限。

格式：chmod 权限参数 文件或目录名

权限参数可以有两种方式：字母表示法和数字表示法。

（1）字母表示法。一个文件用 10 个字符记录文件的权限，第一个字符代表文件类型。“-”表示普通文件；“d”表示目录文件；“b”表示块特别文件；“c”表示字符特别文件。接下来是每 3 个字符代表一类用户的权限。前 33 个是用户主的权限，用 u 代表；中间 3 字符代表和用户主同一个组的权限，用 g 代表；最后 3 字符代表其他用户的权限，用 o 代表；所有用户用 a 表示。每一种用户的权限直接用 r、w、x 代表对文件可读、可写、可执行，然后再用+、-或=将各类用户代表符号 u、g、o 和 r、w、x 三个字母链接起来即可。

例如：

$chmod　u+rwx　f1

用户主对 f1 可以进行读写执行的操作。

$chmod　a=rwx　f1

所有用户对 f1 可进行读写执行的操作。

$ chmod ug+w，o-x text

即设定文件 text 的属性为：用户主（u）增加写权限，与用户主同组用户（g）增加写权限，其他用户（o）删除执行权限。

（2）数字表示法。数字表示法是用 3 位数字 XXX，最大值为 777 来表示的。第一个数字代表用户主存取权限，第二个数字代表同组用户使用权限，第三个数字代表其他用户

存取权限。前面介绍的可读权限 r 用数字 4 表示，可写权限 w 用 2 表示，而可执行权限 x 用 1 表示，即 r=4、w=2、x=1。

假设用户对 f1 的权限是可读、可写、可执行（rwx），用数字表示则把 4、2、1 加起来等于 7，这里 rwx 等价于 4+2+1=7。

至于同组用户和其他用户的权限，顺序指定第 2 位数字和第 3 位数字即可。如果不指定任何权限的话，就要补 0。

例如：

chmod 700 f1

用户主对 f1 的权限是可读、可写、可执行，即 rwx------。

chmod 641 f1

用户主对 f1 的权限是可读、可写，同组用户可读，其他用户可执行。

5.7.4　磁盘存储命令

硬盘空间是一种有限的资源，用户用下面的命令可以随时了解当前硬盘空间的使用情况。

1. df 命令

功能：检查文件系统的磁盘空间占用情况。可以利用该命令获取硬盘被占用了多少空间，目前还剩下多少空间等信息。

格式：df　［选项］

说明：df 命令可显示所有文件系统对 i 节点和磁盘块的使用情况。

该命令中各选项的含义如下：

-a 显示所有文件系统的磁盘使用情况，包括 0 块（block）的文件系统，如/proc 文件系统。

-k 以 k 字节为单位显示。

-i 显示 i 节点信息，而不是磁盘块。

-t 显示各指定类型的文件系统的磁盘空间使用情况。

-x 列出不是某一指定类型文件系统的磁盘空间使用情况（与 t 选项相反）。

-T 显示文件系统类型。

例如：

$df　-k

显示系统所配置的每一个磁盘当前被占用的空间大小。

2. du 命令

功能：显示目录的使用。

格式：du　［选项］[Names…]

说明：该命令逐级进入指定目录的每一个子目录并显示该目录占用文件系统数据块（1024 字节）的情况。若没有给出 Names，则对当前目录进行统计。

该命令的各选项含义如下：

-s 对每个 Names 参数只给出占用的数据块总数。

-a 递归地显示指定目录中各文件及子孙目录中各文件占用的数据块数。若既不指定-s，也不指定-a，则只显示 Names 中的每一个目录及其中各子目录所占的磁盘块数。

-b 以字节为单位列出磁盘空间使用情况（系统缺省以 k 字节为单位）。

-k 以 1024 字节为单位列出磁盘空间使用情况。

-c 最后再加上一个总计（系统缺省设置）。

-l 计算所有的文件大小，对硬链接文件则计算多次。

-x 跳过在不同文件系统上的目录不予统计。

例如：

$du

按块（512 字节）显示一个目录及其所有子目录的使用。

5.7.5 磁盘操作

1. dd 命令

功能：把指定的输入文件复制到指定的输出文件中，并且在复制过程中进行格式转换。可以用该命令实现 DOS 下 diskcopy 命令的作用。先用 dd 命令把软盘上的数据写成硬盘的一个寄存文件，再把这个寄存文件写入第二张软盘，完成 diskcopy 的功能。需要注意的是，应该将硬盘上的寄存文件用 rm 命令删除掉。系统默认使用标准输入文件和标准输出文件。

格式：dd [选项]

例 5：把 net.i 这个文件写入软盘中，并设定读/写缓冲区的数目。注意：软盘中的内容会被完全覆盖掉。

$ dd if = net.i of = /dev/fd0 bs = 16384

例 6：将文件 sfile 复制到文件 dfile 中。

$ dd if=sfile of=dfile

2. fdformat 命令

软盘是用户常用的存储介质之一。软盘在使用之前必须先进行格式化操作，然后用 tar、dd、cpio 等命令存储数据，也可以在软盘上建立可安装的文件系统。

功能：低级格式化软盘。

格式：format [-n] device

说明：该命令的作用是对软盘进行格式化。

-n 软盘格式化后不作检验。

device 指定要进行格式化的设备。

5.7.6 常用系统调用

1. 建立文件 create

当用户想把一批信息作为一个文件存放在磁盘上供以后使用时，可用此操作向系统提出建立文件的要求。

功能：创建一个新文件或重写一个已存在的文件。如果系统中不存在指明文件，核心

便以给定的文件名和许可权限方式来创建一个新文件：如果系统中已有同名文件，核心将该文件的长度截短为 0。创建后的文件随之被打开，并返回其文件描述符 fd；若创建失败，则返回-1。

格式：int create (char *pathname,int mode)

其中，pathname 是用户给予新文件的路径名，mode 为允许对文件进行访问的权限。mode 以二进制形式表示，取其低 9 位，每 3 位为一组，分别表示文件主、同组用户、其他用户对文件的读、写、执行的权限。

2. 打开文件 open

当用户要使用一个已存在的文件时，首先要打开文件，建立用户与文件的联系，把涉及文件的有关目录信息复制到内存中，以便加速系统对文件的检索，为执行文件的读、写等操作做好准备。

格式：int open(char *pathname,int flags);

int open(char *pathname,int flags,mode_ t mode);

open 函数有两个形式。其中，pathname 是要打开的文件名（包含路径名称，缺省时认为在当前路径下面）。flags 是读、写标志，可以是下面的一个值或几个值的组合。

O_RDONLY：以只读方式打开文件。

O_WRONLY：以只写方式打开文件。

O_RDWR：以读写方式打开文件。

O_APPEND：以追加方式打开文件。

O_CREAT：创建一个文件。

O_EXEC：如果使用了 O_CREAT 而且文件已经存在，就会发生一个错误。

O_NOBLOCK：以非阻塞方式打开一个文件。

O_TRUNC：如果文件已经存在，则删除文件的内容。

前面 3 个标志只能使用任意的一个。如果使用了 O_CREATE 标志，就要使用 open 的第二种形式。还要指定 mode 标志，用来表示文件的访问权限。mode 也可以用数字来代表各位的标志。Linux 总共用 5 个数字来表示文件的各种权限。00000 中第一位表示设置用户 ID，第二位表示设置组 ID，第三位表示用户自己的权限位，第四位表示组的权限，最后一位表示其他人的权限。每个数字可以取 1（执行权限）、2（写权限）、4（读权限）、0（什么也没有）或是这几个值的和。

例如要创建一个用户读写执行，组没有权限，其他人读执行的文件。设置用户 ID 位可以使用的模式是 1（设置用户 ID）0（组没有设置）7（1+2+4）0（没有权限，使用缺省）5（1+4）即 10705。

open("temp",O_CREAT,10705);

如果打开文件成功，open 会返回一个文件描述符，以后对文件的所有操作就可以对这个文件描述符进行。

3. 关闭文件 close

当不再使用一个已打开的文件时应将文件关闭，以切断用户与该文件的联系。由于文

件可被多个进程共享，故只有再无其他进程需要此文件时，对索引节点中的 i.count 进行减 1 后，若值为 0，才能关闭该文件。

格式：int close(fd)；

close 处理程序根据文件描述符 fd 找到进程打开文件表表项，从相应的用户文件描述符表项中获得它指向文件表项的指针 fp，再对该文件表项中的 f.count 和 i.count 做减 1 操作，若 f.count=0 则置该表项为空闲，若 i.count=0 则置该活动索引节点为空闲，把 fd 所指的进程打开文件表表项清 0。

4. 读文件 read

打开文件后，就要对文件进行读取了，可以调用函数 read 进行文件的读操作。

格式：int read(int fd, void *buffer,ntypes)

其中，fd 是读操作的文件描述符，buffer 是所读的文件信息存放地址，nbytes 是要读的字节数。

对于普通的文件，read 从指定的文件中读取 nbytes 字节到 buffer 缓冲区中（必须提供一个足够大的缓冲区），同时返回 nbytes。

如果 read 读到了文件的结尾或被一个信号所中断，返回值会小于 nbytes，如果是由信号中断引起返回，而且没有返回数据，read 会返回-1。当程序读到了文件结尾的时候，read 会返回 0。

5. 写文件 write

当用户要向文件增加记录时，可通过 write 命令执行写操作。

格式：int write(int fd, void *buffer,nbytes)

其中，fd 是要进行写操作的文件描述符，buffer 是要写入文件内容的内存地址，nbytes 是要写的字节数.

write 从 buffer 中写 nbytes 字节到文件 fd 中，成功时返回实际所写的字节数，若出错为-1。

其返回值通常与参数 nbytes 的值不同，否则表示出错。write 出错的一个常见原因是磁盘已写满，或者超过了对一个给定进程的文件长度限制。对于普通文件，写操作从文件的当前位移量处开始。如果在打开该文件时，指定了 O_APPEND 选择项，则在每次写操作之前，将文件位移量设置在文件的当前结尾处。在一次成功写之后，该文件位移量增加实际写的字节数。

6. lseek 函数

每个打开文件都有一个与其相关联的“当前文件位移量”，它是一个非负整数，用以度量从文件开始处计算的字节数。通常，读、写操作都从当前文件位移量处开始，并使位移量增加所读或写的字节数。按系统默认，当打开一个文件时，除非指定 O_APPEND 选择项，否则该位移量被设置为 0。

可以调用 lseek 显式地定位一个打开文件。

格式：off_t　lseek(int fd,off_t offset,int whence);

返回：若成功则返回新的文件位移，若出错返回-1。

对参数 offset 的解释与参数 whence 的值有关。

若 whence 是 SEEK_SET，则将该文件的位移量设置为距文件开始处 offset 个字节。若 whence 是 SEEK_CUR，则将该文件的位移量设置为其当前值加 offset，offset 可为正或负。若 whence 是 SEEK_END，则将该文件的位移量设置为文件长度加 offset，offset 可为正或负。

若 lseek 成功执行，则返回新的文件位移量，为此可以用下列方式确定一个打开文件的当前位移量：off_t currpos;

currpos=lseek(fd,0,SEEK_CUR);

这种方法也可用来确定所涉及的文件是否可以设置位移量。如果文件描述符引用的是一个管道或 FIFO，则 lseek 返回-1，并将 errno 设置为 EPIPE。

本章小结

本章首先介绍了文件与文件系统的概念。文件是计算机存储信息的基本单位，是一组相关记录的集合。文件系统是操作系统中负责存取和管理文件信息的机构，是用户接触、使用操作系统过程中面对的部分，它负责管理静态的文件。

文件的结构分为逻辑结构和物理结构。文件的逻辑结构分为有结构的记录式文件和无结构的字符流式文件。Linux 系统采用无结构的字符流式文件。

文件的物理结构是文件在外存上的存储组织形式，主要涉及文件的信息如何存放在磁盘上。文件的物理组织形式有连续文件结构、串联文件结构、索引结构、多重索引结构。连续文件结构是把逻辑文件中的记录顺序地存储到连续的物理盘块中，优点是管理简单，顺序存取速度快，但是存取时必须事先知道文件的长度，文件存储时必须占用连续的存储空间，从而产生很多外零头。串联文件把逻辑文件存储到外存时，不要求占用连续的空间，而是存放在离散的多个盘块中，然后通过连接指针将这些离散的盘块连接成一个链，这种组织形式解决了连续文件中产生外零头的问题，同时增、删、改文件也方便。索引文件结构是另一种非连续分配的文件存储结构，通过索引表实现逻辑地址与物理地址的映射关系，但当文件太长时，索引表会很长，既占用大量存储空间，又影响检索速度。多重索引结构可以缩短索引表的长度，但会增加访问磁盘的次数。在 Linux 系统中文件使用混合索引的方式。

文件目录是用来组织文件和检索文件的关键数据结构。文件目录有单级、二级及树形目录 3 种形式。单级目录最简单，但存在文件重名问题。二级目录为各个用户单独建立一个目录，每个用户的文件都存储在自己的目录下。对二级目录进行扩展，成为树形文件目录结构，树形目录下允许用户创建自己的子目录，方便用户更合理地组织文件。在 Linux 系统中采用的就是带链接的树形目录结构，它利于实现对文件或目录的共享。

当创建文件或扩充文件时，需要申请磁盘空间；删除文件时需要回收磁盘空间，因此对文件存储空间的管理也是一个重要问题，主要管理方式有空闲空间表法、空闲块链、位示图、空闲块成组连接法。在 Linux 系统中采用的就是成组连接法。

文件的共享和安全性是文件系统的一个重要问题。实现共享常用的方法有绕弯路法和

连接法。对文件存取控制是和文件共享、保护紧密相连的。存取控制可采用口令、密码、存取控制矩阵、存取控制表等方法来限制用户对文件的权限，达到保护文件的目的。

Linux 系统有很多优点，它提供了一整套对文件、目录、文件系统等的操作命令，如创建及删除文件和目录、复制和显示文件、修改文件或目录的权限等。

习 题 五

5-1 什么是文件和文件系统？文件系统有哪些功能？

5-2 Linux 文件可以根据什么分类？可以分为哪几类？各有什么特点？

5-3 什么是文件的逻辑结构？什么是文件的物理结构？Linux 文件系统分别采用什么样的结构？有什么优点和缺点？

5-4 目前广泛使用的目录结构形式是哪种？它有什么优点？

5-5 常见的文件存储空间管理有哪几种？各自有什么特点？Linux 系统的存储空间采用什么方法？

5-6 内存管理和外存管理有何异同？

5-7 为什么要打开文件？叙述在 linux 文件系统，打开文件/home/user01/myfile 的过程？

5-8 在 Linux 系统中，一个盘块大小为 1KB，每个盘块号占 4B，则一个进程要访问一个相对于文件开始的偏移量为 263168B 处的数据时，试计算应直接访问还是索引访问，几级索引？

5-9 有一个 UNIX 文件的权限为 rwxr_x_ _ _，该文件的文件主为 uid=12，gid=1，有一用户 B 的 uid=6，gid=1，问系统是否允许用户 B 执行该文件？为什么？

5-10 说明 Linux 系统磁盘的结构及各部分的功能。

5-11 磁盘容量为 1.2GB 的硬盘，假设盘块大小为 1KB，每个盘块号在 FAT 表区占 4B，作为磁盘的设计人员，应设 FAT 区为多长？

5-12 设在 UNIX 中有一进程 P，P 中有一操作需要访问偏移量为 14000 处的数据，试问 UNIX 如何实现地址变换？

5-13 利用 link 系统调用进行连接时，如果 patr1=/usr/zhang/ff.c，path2=/usr/li/message/ff.c，试画出此时的树形目录图。

5-14 实现文件共享时，如果不先进行连接，会产生什么后果？

第6章　设备管理

本章学习目标

I/O 设备是计算机系统的一个重要组成部分。随着计算机技术的发展和计算机应用的推广，I/O 设备日趋多样化、复杂化和智能化。由于 I/O 设备种类繁多，它们的特性和操作方式差别很大，使得设备管理成为操作系统中最繁杂且与硬件关系最密切的部分。

通过本章的学习，读者应该掌握以下内容：

- 掌握数据的传输方式
- 掌握中断技术的概念及流程
- 掌握缓冲技术的原因及原理
- 理解设备分配技术
- 掌握设备独立性的概念
- 了解设备驱动的过程

6.1　I/O 系统组成

I/O 系统是计算机系统中完成数据输入、输出的子系统，它包括输入/输出设备、相应于各设备的设备控制器，在大中型计算机中，还配置了 I/O 通道。

6.1.1　I/O 设备

I/O 设备是人们使用计算机的途径，不同的使用方式需要配备不同的设备，如输入文字一般用键盘，但在某些应用场合，需要扫描仪，输入声音需要话筒，打印信息需要打印机，拨号上网需要调制解调器，实时控制领域需要传感器。人们应用计算机的方式多种多样，所以计算机系统中可以配置的 I/O 设备种类繁多。设备的应用领域不同，其物理特性各异，但某些设备之间具有共性，为了简化对设备的管理，可对设备分类，或对同类设备采用相同的管理策略。

从不同的角度可以对设备进行不同的分类。

1. 按使用特性分类

（1）存储设备。存储设备又称为外存、后备存储器、辅助存储器，用于永久保存用户需要的信息（程序和数据）。外存与内存一起构成计算机系统的存储体系，内存只用于暂时存储当前要处理的信息，而外存用于永久保存用户要用计算机来处理的信息。外存比内存容量大，

存储单元价格低。内存的介质是半导体电子元件，外存的介质是磁盘、磁带、光盘等。

（2）输入/输出设备。这是用户直接使用的设备，用户通过直接操作输入/输出设备与计算机通信。输入设备是计算机用来接收外部世界信息的设备，如用户从键盘输入命令或数据，从扫描仪输入图像。输出设备是将计算机加工处理好的信息送向外部世界的设备，如显示器、打印机、音箱。

2. 按传输速率分类

（1）低速设备。传输速率为每秒钟几个字节到几百字节。典型的低速设备有键盘、鼠标、低速打印机等。

（2）中速设备。传输速率在每秒种数千个字节到每秒钟数万字节之间。典型的中速设备有激光打印机等。

（3）高速设备。传输速率在每秒种数百千字节到每秒钟数十兆字节之间。典型的高速设备有磁带机、磁盘驱动器、光盘驱动器等。

3. 按信息传输单位分类

（1）块设备。数据传输以数据块为单位进行的设备称为块设备，如磁盘驱动器、磁带机等高速外存储器等，这类设备用于存储信息。典型的块设备是磁盘驱动器。磁盘是圆形的存储介质，其存储空间的划分方式是从圆心出发将盘面划分为若干半径不同的同心圆，每个圆称为一个磁道，然后将每个磁道等分为若干个弧段，每个弧段称为一个扇区。磁盘驱动器工作时以盘块作为数据传输单位，每个盘块包含整数个扇区。为了能对盘块寻址，给每个盘块规定了惟一的地址。硬盘一般由若干个盘片叠放在一个圆轴上构成，在同一位置上的磁道称为柱面。

（2）字符设备。每次传输数据以字节为单位的设备称为字符设备，如打印机、显示器、键盘。字符设备即上面所说的输入/输出设备，用户通过这些设备使用计算机。由于字符设备直接和用户打交道，应用于各种场合，所以其种类繁多。字符设备以字符流的方式传送数据，每个传输单位——字符是不可寻址的。

4. 按资源分配方式分类

（1）独占设备。当有多个并发进程共享某一设备时，只允许一个进程访问的设备称为独占设备，换句话说，独占设备是不允许两个以上的并发进程交替访问的设备。如果有多个并发进程要共享该类设备，则应互斥使用，即将该类设备分配给一个进程后，便由该进程独占，在其占有期间，即使设备空闲，也不允许其他设备使用，直至其主动释放该设备后，其他进程才可使用，打印机就是一种独占设备，如果两个进程交替使用打印机，则会使两个进程的打印内容掺杂在一起，没法区分，使打印的内容失去了本来的意义。独占设备的利用率较低。

（2）共享设备。允许多个并发进程交替使用的设备称为共享设备，该类设备没有占有权的问题，只要设备空闲就可使用。典型的共享设备是磁盘机。

（3）虚拟设备。针对独占设备使用不方便及利用率低的问题，在并发进程使用独占设备时，可以通过虚拟技术把一台独占设备变换为若干台逻辑设备，每个进程使用一台逻辑设备。例如在多个并发进程共享一台打印机时，可以为每个进程分配一个打印缓冲区，

在打印时不直接输出到打印机，而是先输出到打印缓冲区，然后在操作系统的管理下，在适当时机真正在打印机上打印。在此，打印缓冲区就是一台逻辑打印机，即虚拟打印机。

6.1.2　设备控制器

设备控制器是 CPU 和 I/O 设备之间的接口，它接收从 CPU 发来的命令，并控制 I/O 设备工作，并向 CPU 发送中断信号。

一个设备控制器可以交替控制几台同类设备，与同一设备控制器相连的每台设备都有惟一的地址。在微型机和小型机中，设备控制器常做成印刷电路卡的形式，因而也称为接口卡，可将其插入计算机主板上的 I/O 插槽中。

1. 设备控制器的功能

（1）接收并识别来自 CPU 的命令。如同 CPU 的指令系统，设备控制器也有自己的一套“指令系统”，设备控制器能执行一组特定的指令，分别完成不同功能。在设备控制器中应该配置控制寄存器，用来存放接收的命令和参数。例如，磁盘控制器可以执行 read、write、format 等命令，有些命令带有参数。

（2）数据传输。实现主机（CPU、内存）与控制器、控制器与设备之间的数据传输。对于前者，是通过数据总线进行；对于后者，设备将数据送入控制器，或从控制器送到设备。为此，在设备控制器中必须设置数据寄存器。

（3）记录设备的状态。设备状态影响对设备的访问，而 CPU 不能直接访问设备，所以，控制器要记录设备的状态。例如，当设备处于发送就绪状态时，CPU 才能启动控制器从设备中读出数据。为此，在控制器中应该设置一个状态寄存器，保存设备的当前状态，CPU 可以通过访问该寄存器来了解设备的状态。

（4）识别设备地址和寄存器地址。一个设备控制器可以连接控制多台同类设备，为了能单独访问某一设备，系统中的每个设备都要有惟一的地址，就像内存中的每一个单元都有一个地址一样。

此外，为了使 CPU 能访问设备控制器中的端口（寄存器），这些端口应该具有惟一的地址，控制器要能够识别这些地址。为此，在设备控制器中应该配置地址译码器。

（5）差错控制。为保证数据输入的正确性，设备控制器要对由输入设备传送来的数据进行差错检测。若发现传送中出现错误，则将差错检测码置位，并向 CPU 报告，CPU 将本次传送来的数据作废，并重新进行一次传送。

2. 设备控制器的组成

现有的大多数设备控制器由以下 3 部分组成，如图 6-1 所示。

（1）设备控制器与 CPU 的接口。该接口用于实现设备控制器与 CPU 之间的通信，该接口中有 3 类信号线：数据线、地址线、控制线。数据线通常与两类寄存器相连接。

1）数据寄存器。设备控制器中可以有一个或多个数据寄存器，用于存放从设备送来的数据（输入）或从 CPU 送来的数据（输出）。

2）控制/状态寄存器。控制器中同样可以有一个或多个控制/状态寄存器，用于存放从 CPU 送来的控制信息或由设备产生的状态信息。

（2）设备控制器与设备的接口。在一个设备控制器上，可以连接一台或多台设备。相应地，在控制器中有一个或多个设备接口，一个接口连接一台设备，每个接口中都有数据、控制和状态 3 种类型的信号。

（3）I/O 逻辑。设备控制器的核心部分是对设备进行具体控制的 I/O 逻辑。I/O 逻辑部分对接收的来自 CPU 的 I/O 命令字进行译码，然后通过与设备的接口向设备发出控制信号。

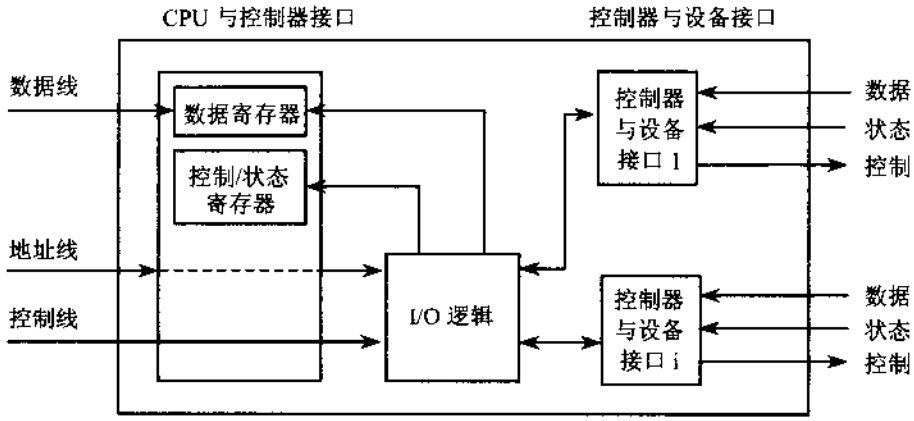


图 6-1 设备控制器的构成

6.1.3 I/O 通道

I/O 通道是一种专门负责 I/O 操作的小型处理机，它接受 CPU 的命令，独立管理 I/O 操作过程，实现内存和设备之间的成批数据传输。通道相当于一个协处理器，类似于以前微机中配置的数学协处理器。通道有自己的一套简单的指令系统，可以构成通道程序，通过独立执行通道程序来完成 CPU 交付的 I/O 操作，如图 6-2 所示。有了通道后，如果当前正运行的进程要进行 I/O 操作，则不需要由 CPU 执行完成 I/O 操作的程序，即不需要由 CPU 直接和设备控制器通信，具体控制 I/O 操作过程，而只需向 I/O 通道发出 I/O 指令，通道接受该指令后，就会从内存中取出完成本次 I/O 操作要执行的通道程序，然后由通道执行通道程序，具体控制 I/O 操作过程。当通道完成了规定的 I/O 任务后，向 CPU 发出中断信号，由 CPU 决定后续操作。

通道的引入使 CPU 可以“专心”进行计算，而把 I/O 任务完全交给通道完成，二者各负其责。当然，CPU 具有最高控制权，通道要接受 CPU 的控制，在 CPU 的统一指挥下工作。二者的分工使计算和 I/O 操作可以并行，即 CPU 和设备并行工作，从而提高了资源利用率。

一个通道可以同时连接若干个设备控制器，每个设备控制器又可同时连接若干同类设备。当然，一个通道不可能同时控制它连接的所有设备控制器，而是交替控制它们。同样，一个设备控制器也不能同时控制它所连接的所有设备，而是交替控制。

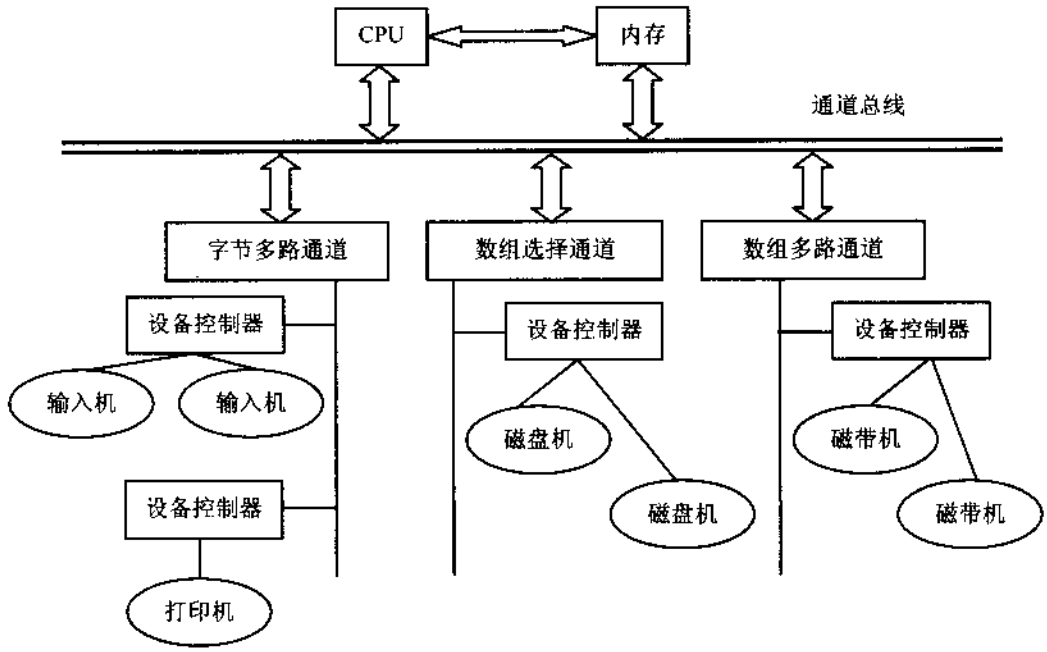


图 6-2　具有通道的 I/O 系统结构

由于外部设备种类繁多，各有特性，因此通道也有多种类型。根据信息交换方式的不同，把通道分为 3 种。

（1）字节多路通道。字节多路通道以字节为单位传输数据，它用于连接低速字符设备，如终端、串行打印机、卡片机。字节多路通道可以连接若干 I/O 设备，通过按时间片轮转方式分时执行多个通道程序来控制 I/O 设备进行 I/O 操作。当一个通道程序控制某台设备传送一个字节后，通道就转去执行另一个通道程序，控制另一台设备传送一个字节，依次类推。

（2）数组选择通道。数组选择通道以数组（由多个字节构成）为单位传输数据，从而数据传输速率较高，通常连接高速设备，如磁盘机。数组选择通道以串行方式执行通道程序，即每次执行一个通道程序，控制一台设备进行 I/O 操作，直至此通道程序运行完毕，才执行另一个通道程序，控制另一台设备进行 I/O 操作。当通道分配给某台设备后，便一直由该设备独占，由于通道运行速率高于设备，故在设备工作时，通道有一段等待时间，但即使通道闲置，也不允许其他设备使用该通道，直至该设备传送完毕释放该通道，可见，这种通道的利用率低。

（3）数组多路通道。数组多路通道以数组为单位传输数据，通常连接中高速设备，如磁带机。数组多路通道其实是将字节多路通道的数据传输单位改成了数组，可以同时获取较高的数据传输速率和通道利用率。数组多路通道以分时方式并发执行多个通道程序，每次执行一个通道程序的一条指令后就转向执行另一个通道程序，每条通道指令可以控制传送一组数据。

6.2 数据传输控制方式

计算机是一个信息处理工具，人们将要处理的信息首先输入到内存中，然后由 CPU 执行程序对其进行处理，再将处理结果存储到内存中，最后送到输出设备，这便是一个完整的数据传输过程，在此过程中，内存为核心。从计算机的整体工作流程（输入→处理→输出）看，数据传输指的是内存和 I/O 设备之间的数据传输，更具体地说，是进程的数据存储区与 I/O 设备之间的数据传输，如图 6-3 所示。

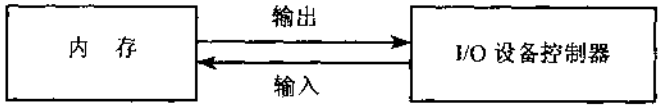


图 6-3 内存和设备控制器之间的数据传输

实际进行内存和 I/O 设备之间的数据传输时，由于某些原因，不一定直接在内存和 I/O 设备之间进行，可以通过一些中间途径，如通过 CPU 的寄存器做中转，Intel 8086 指令系统中的输入、输出指令 IN、OUT 就是在设备控制器端口（寄存器）和 CPU 的累加寄存器之间进行数据传输。

6.2.1 程序直接控制方式

由程序直接控制内存与 I/O 设备之间的数据传输称为“忙等”方式或循环测试方式，即当要在内存和 I/O 设备之间进行信息传输时，由 CPU 向相应的设备控制器发出命令，由设备控制器控制 I/O 设备进行实际操作。在 I/O 设备工作时，CPU 执行一段循环测试程序，不断测试 I/O 设备的完成状况，以决定是否继续传输下一个数据。若设备未完成此次数据传输，则继续测试，若设备完成了此次数据传输，则进行下一次数据传输或继续执行程序。程序直接控制方式进行数据传输的方式如图 6-4 所示。

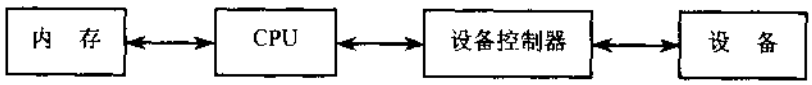


图 6-4 程序直接控制方式进行数据传输

由上图可以看出，数据传输单位是设备控制器的数据缓冲寄存器的容量（字节数），用户程序要求的一个数据传输可能需要传输多次。

输出数据时的工作过程如下：

（1）将需要输出的数据由 CPU 中的数据寄存器送到相应设备控制器的数据缓冲寄存器中。

（2）把一个启动位为“1”的控制字写入该设备的控制寄存器，启动该设备工作。

（3）测试状态寄存器中的完成位，若为 0，继续测试，若为 1，则进行下一次数据传输或继续执行程序。

程序直接控制方式的缺点是 CPU 和设备利用率低，从而导致计算机工作效率低，原因在于 CPU 和设备只能串行工作。在 CPU 工作时，设备处于空闲状态，在设备工作时，CPU 处于空闲状态（忙等未做有效工作，可以看作空闲）。

6.2.2　中断控制方式

中断控制方式使 CPU 与外设并行工作，从而提高 CPU 和设备的利用率，最终提高计算机系统的工作效率。

当要在主机和 I/O 设备之间进行信息传输时，由 CPU 向相应的设备控制器发出命令，由设备控制器控制 I/O 设备进行实际操作，每次的数据传输单位是设备控制器的数据缓冲寄存器的容量。I/O 设备工作时，相应进程放弃处理机，处于等待状态，由操作系统调度其他就绪进程占用 CPU。I/O 操作完成时，由设备控制器向 CPU 发出中断信号，通知 CPU 本次 I/O 操作完成，然后由 CPU 执行一个中断处理程序，对此情况做出相应反应。中断处理过程一般首先保护现场；然后将等待 I/O 操作完成的进程唤醒，使其进入就绪状态；最后转进程调度。

中断控制方式的缺点是由于每次的数据传输单位是设备控制器的数据缓冲寄存器的容量，单位传输数据量较小，进程每次需要传输的数据被分为若干部分进行传输，中断次数很多，每次中断都要运行一个中断处理程序，耗费 CPU 的时间很多，使 CPU 的有效计算时间减少。

6.2.3　DMA 方式

DMA 方式减少中断次数，提高 CPU 利用率。

减少中断次数，有如下两种方法：

（1）增大数据缓冲寄存器的容量。

（2）另外配置一个数据传输控制器件，进行成批数据传输。

由于方法（1）因各方面的原因很难实施，在此采用方法（2），具体方案是配置一个 DMA 控制器。DMA（Direct Memory Access）直接内存访问的基本思想是，当要在内存和 I/O 设备之间进行大批量的数据传输时，不由 CPU 控制具体传输过程，CPU 只需提出要求，由 DMA 控制器控制具体传输过程。在 DMA 控制器的控制下，在外设和内存之间开辟了直接的数据传输通路，不需要经过 CPU 的数据寄存器中转。

DMA 方式的数据传输单位是数据块，仅在数据块传输结束时才向 CPU 发出中断信号，从而减少了中断次数。

DMA 方式下的数据输入过程如下：

（1）当某一进程要求设备输入数据时，CPU 把准备存放输入数据的内存始地址和要传送的字节数分别送入 DMA 控制器中的内存地址寄存器和传送字节计数器。

（2）启动 DMA 控制器，控制设备输入数据。

（3）该进程进入等待状态，等待数据输入的完成，操作系统调度其他就绪进程占用CPU。

（4）输入完成时，DMA 控制器发出中断信号，CPU 响应后进行相应的中断处理。

DMA 方式的缺点是智能化程度较低，CPU 干预较多，因为仍要由 CPU 运行设备驱动程序来进行。

6.2.4 通道控制方式

为了提高计算机系统的运行效率，就要使 CPU 摆脱繁忙的 I/O 事务，而配置专门处理 I/O 事务的机构——通道。通道是一个专门进行 I/O 操作的处理机，它接受主机的命令，独立地执行通道程序，对外部设备的 I/O 操作进行控制，在内存和外设之间直接进行数据传送。当主机交付的 I/O 任务完成后，通道向中央处理机发出中断信号，请求 CPU 处理。

I/O 通道与中央处理机不同，主要表现在以下两个方面，一是其指令类型单一，这是由于通道硬件比较简单，所能执行的命令主要局限于与 I/O 操作有关的指令；再就是通道没有自己的内存，通道所执行的通道程序是放在主机内存中的，通道与 CPU 共享内存。

1. 通道命令

通常，计算机系统的通道具有以下 3 类基本操作：

（1）数据传送类，如读、写、反读、断定（检验设备状态）。

（2）设备控制类，如控制换页、磁带反绕等。

（3）转移类，即通道程序内部的控制转移。

例如，某系统中通道命令的格式如下表所示。

0 7	8 31	32 47	48 63
操作码	数据主存始址	特征	字节计数

每条通道命令由 4 个字段组成。

（1）操作码（0～7）：表示通道要执行的命令。

（2）数据主存始址（8～31）：表示本命令要访问的主存数据区的起始地址。

（3）特征位（32～47）：只使用了 32～36 位，其余未用。37～39 位必须置 0，否则认为通道命令错。

特征位进一步规定了本通道命令的意义，其各特征如下：

数据链特征（32 位）：当该位为 1 时，在执行本命令后，使计数值为 0 并取下一条通道命令（CCW），下一条 CCW（它规定了新的数据主存始址、特征和计数）继续执行原来的操作。

命令链特征（33 位）：该位为 1 时，当前命令正常结束后，顺序执行下一条 CCW。在通道程序中，除最后一条 CCW 外，每条 CCW 都应指定 32 位为 1，或 33 位为 1。最后一条 CCW 的所有特征位为 0 时，表示通道在完成当前命令后自动停止。

禁发长度错特征（34 位）：表示对不正确的数据长度不发出错信号。一般地说，如果通道命令给出的字节数与 I/O 设备所用的字节数不一致，就发出错信号。如果该位置为 1，

则禁止发出出错信号。

跳读特征（35 位）：表示禁止数据传送到内存。当该位为 1 时，从 I/O 设备上读出的数据不写入内存，这样就可以跳过一些数据。

程序控制中断特征（36 位）：如果该位为 1，只要取出 CCW 并开始操作，则通道就向 CPU 发出中断信号。

（4）字节计数（48～63 位）：规定了数据区的字节数，数据主存始址和字节计数这两个字段主要用于数据传送类命令。

2. 通道地址字和通道状态字

在通道进行输入/输出操作期间，要访问两个固定内存单元：通道地址字（CAW）和通道状态字（CSW）。

通道地址字的格式如下表所示。

0 3	4 7	8 31
钥	0000	通道程序首地址

通道状态字的格式如下表所示。

0 3	4 7	8 31	32 39	40 47	48 63
钥	0000	下一条 CCW 地址	设备状态	通道状态	剩余计数

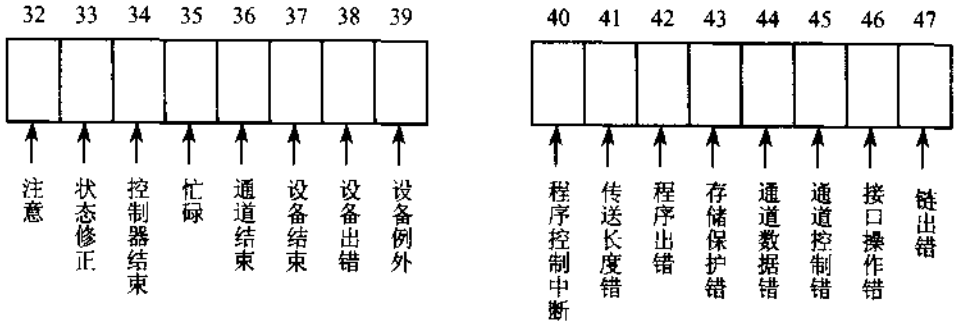
其中：

钥——通道访问主存时所用的保护钥，它由 CAW 的前 4 位送来。

下一条 CCW 地址——通道命令计数，每当一条 CCW 执行完毕，该地址自动加 8，指向下一条 CCW。

设备状态——记录设备状态的字节（如图 6-5（a）所示）。

通道状态——记录通道状态的字节（如图 6-5（b）所示）。



（a）设备状态字节　　（b）通道状态字节

图 6-5　设备状态字节和通道状态字节

剩余计数——上一条 CCW 执行结束后，剩余数据的字节数。

3. CPU 和通道间的通讯

（1）输入/输出指令。某系统中输入/输出类指令主要有 SIO、TIO、HIO、TCH。

SIO（START I/O）：用于启动指定通道和设备开始工作，SIO 指令发出后，根据通道和设备状态建立结果特征，即条件码。CPU 根据执行结果建立的条件码得知此次启动是否成功，再确定下一步的动作。

TIO（TEST I/O）：用以检查 I/O 操作是否完成，建立相应的条件码。

HIO（HALT I/O）：强行停止通道和 I/O 设备正在执行的操作。

TCH（TEST CHANNEL）：测试通道，根据对通道测试的结果建立相应的条件码。

（2）CPU 和通道间的通讯。CPU 通过上述一些输入/输出指令控制通道，通道执行通道程序完成 CPU 交付的任务，当通道完成任务后，向 CPU 发送中断信号，CPU 进行相应的中断处理。具体过程如图 6-6 所示。

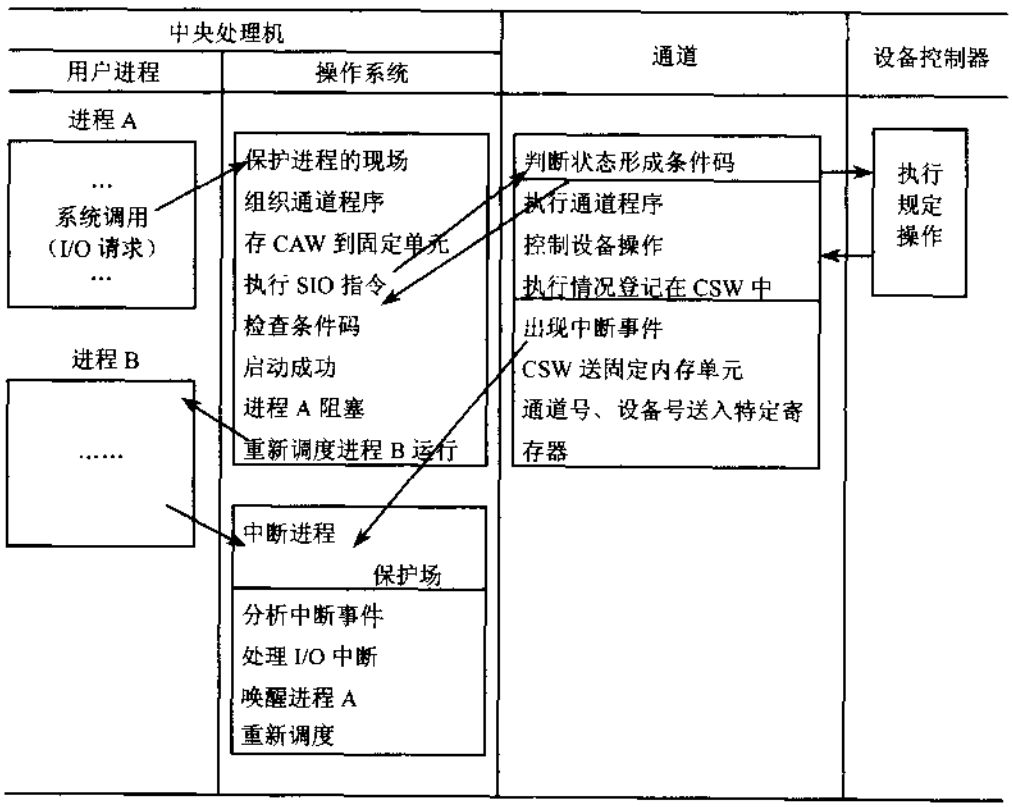


图 6-6 CPU 和通道间的通讯

6.3 中断技术

6.3.1 中断的概念

在多任务计算机系统中，多个进程并发执行，各进程要完成自己的任务必须获得 CPU

的控制权。在分时系统中，采用分时共享 CPU 方式对并发进程进行调度，使它们能公平地使用 CPU，从而完成各自的任务。各并发进程被轮流分配给一定的时间片，当某进程的时间片用完时，操作系统暂停该进程的运行，将处理机分配给下一个进程运行，该进程也运行一个时间片，依次类推，轮完一遍后，再开始新一轮时间片的轮转。要实现这种分时共享 CPU 的方式，关键是要准确把握各进程的运行时间，要能定时。可以采用硬件定时器来定时，预先设置定时器，当时间片用完时便向操作系统发送一个时钟中断信号，通知操作系统当前运行进程的时间片用完，操作系统便可调度另一个进程运行，此时采用了时钟中断技术。

当进程进行输入输出操作时，如果主机采用查询方式来等待慢速外设操作的完成，将会浪费 CPU 资源，降低计算机系统的工作效率。为了使处理机和外设能够并行工作，现代计算机系统都引入了中断技术。

计算机系统还具有自动处理各种事故的能力，如电源故障、地址越界等。

中断是指处理机在执行进程的过程中，由于某些事件的出现，中止当前进程的运行，转而去处理出现的事件，待处理完毕后返回原来被中断处继续执行或调度其他进程执行。

6.3.2　中断源

引起中断的事件称为中断源。计算机中的中断源种类很多，大致可分为如下几类。

1. 强迫性中断源

（1）硬件故障。如电源故障、主存储器故障等。

（2）程序性错误。由执行机器指令引起的错误，如除数为零、操作数溢出、非法指令、目态下使用特权指令、地址越界等。

（3）外部事件。时钟中断、重启动中断等。

（4）I/O 中断事件。外设完成 I/O 操作或 I/O 操作出错，如打印完成、打印缺纸等。

2. 自愿性中断源

进程执行访管指令请求 OS 服务，如请求分配外设、请求 I/O 等。由访管指令引起的中断称为访管中断。

6.3.3　中断响应

中断源向 CPU 发出请求中断处理信号称为中断请求。当中央处理机发现有中断请求信号时，中止当前程序的执行，并自动进入相应的中断处理程序的过程称为中断响应。中断响应由硬件中断机构完成。

1. 中断响应的过程

（1）保护被中断进程的现场。为了以后能从断点处继续执行被中断的进程，系统必须保存当前程序状态字 PSW 和程序计数器 PC 的值。

（2）分析中断原因，转入相应的中断处理子程序。只要将中断处理程序的入口地址送入程序计数器，将程序状态字送入程序状态字寄存器，便转入了中断处理程序。

2. 程序状态字

任何程序运行时都有反映其运行状态的一组信息，有的机器将这些信息集中存放在CPU中的一个寄存器中，称为程序状态字（PSW，Program Status Word），存放这些信息的寄存器称为程序状态字寄存器。并不是所有的机器都这样做，有的机器采用分散存放的方法。

程序状态字是反映程序执行时机器所处的现行状态的代码。它的主要内容包括：

（1）处理器的当前执行状态：目态、管态。

（2）当前指令的执行情况：是否进位、溢出、为零。

6.3.4 中断处理

中断响应完成后，进入中断处理过程。中断处理是由中断处理程序完成的。

中断源类型繁多，对不同的中断源，处理方式也不同，因此针对不同的中断源要分别编写不同的中断处理程序。下面简要介绍对各种中断源的处理过程。

1. 硬件故障中断的处理

机器发生硬件故障后，往往需要由人工干预去排除故障。操作系统所做的工作一般只是保护现场，防止故障蔓延，向操作员报告故障信息。

（1）电源故障的处理。当电源发生故障，例如掉电（指电源电压持续下降）时，硬件设备能继续正常工作几个毫秒，操作系统利用这几个毫秒的时间可以做好以下3项工作：

一是将当前现场保存到非易失性存储器中，以便电源恢复正常后继续往下执行被中断了的程序。非易失性存储器即使在电源去除后也能保留存入的数据。

二是停止外部设备工作。

三是停止处理器工作，使整个系统既不响应中断也不继续执行指令，处于等待状态。

当故障排除后，操作员从一个约定点重新启动操作系统，恢复系统工作。

（2）内存故障处理。内存的奇偶校验装置发现内存读写错误时就产生中断。操作系统首先停止涉及的程序的运行，然后向操作员报告出错单元的地址和错误性质（处理器访问内存错还是通道访问内存错）。

2. 程序性中断的处理

处理程序性中断有两种办法。对于那些纯属程序错误而又难以克服的事件，例如地址越界、非管态时执行了管态指令、企图写入半固定存储器或禁写区等，操作系统只能将出错程序的名字、出错地点和错误性质报告发送给操作员，请求干预，对于其他一些程序性中断事件，如溢出、跟踪等，不同的用户往往有不同的处理要求，所以操作系统可以将这些程序性中断事件交给用户程序处理，这时要求用户编制处理中断事件的程序。

3. 外部中断的处理

下面以时钟中断处理过程为例，说明外部中断的处理过程。

计算机中有一个定时器，使用定时器时，用一条特殊指令将指定值写入定时器，然后

启动定时器工作。定时器开始工作后，每隔一定时间，值减 1，当该寄存器内容为 0 时，发出时钟中断信号。

例如，当某个进程需要延迟一段时间，它可以通过一个系统调用命令发出延迟请求，并将自己挂起，当延迟时间到时，产生时钟中断信号，时钟中断处理程序唤醒被延迟的进程。

4. 输入输出中断的处理

输入输出中断一般可分为传输结束中断、传输错误中断和设备故障中断。

（1）传输结束中断处理。置设备及相应的控制器为空闲状态。

（2）传输错误中断处理。置设备及相应的控制器为空闲状态；报告传输错误；若设备允许重复执行，则重新组织传输。

（3）设备故障中断处理。置设备及相应的控制器为空闲状态，报告设备故障。

5. 访管中断的处理

当用户进程请求操作系统服务时，通过访管指令进行系统调用，中断机构根据访管指令中的参数转入相应的系统功能程序。

6.4　缓冲技术

缓冲是指通信双方不直接通信，而是通过一个缓冲器中转，如图 6-7 所示。

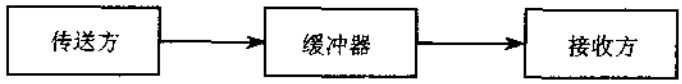


图 6-7　缓冲器

缓冲器是一个存储器，它可以是硬件级的，即独立于内存外设置专门的硬件缓冲器；也可以是软件级的，即由软件在内存中开辟一块缓冲区域。硬件缓冲器增加了计算机的制造成本，所以除了在关键的地方采用硬件缓冲器外，大都采用软件缓冲区。

6.4.1　缓冲的引入

引入缓冲技术的原因如下：

（1）减少读块设备的次数。当进程在块设备上进行读操作（输入）时，通过系统调用，由操作系统将数据读入缓冲器，然后送入相应进程的数据存储区，由进程处理。缓冲器内保存的信息可能会被再次读到，因此，当再次对此块设备进行读操作时，可以先查看一下所读信息是否在缓冲器内，若在缓冲器内，则可以直接从缓冲器内读取，而不必从设备读取。这种场合要求对缓冲器的读取速度快于对块设备的读取速度，则当命中率（所读数据在缓冲器内的次数与读操作的总次数的比值）较高时，会明显提高读操作的速度。CPU 和内存之间的高速缓存的作用就是这样的。

（2）减少对 CPU 的中断次数，放宽对中断响应时间的限制。这是硬件缓冲器的作用。

例如，在远程通信系统中，如果从远程终端发来的数据仅用一位缓冲来接收，则必须每收到一位数据便中断一次 CPU。这样，对于速率为 9.6kb/s 的数据通信而言，就意味着中断 CPU 的频率也为 9.6kb/s，即每 100 微秒就中断 CPU 一次，而且 CPU 必须在 100 微秒内响应，否则缓冲区内的数据将被覆盖。若设置一个具有 8 位的缓冲寄存器，则可使 CPU 被中断的频率降低为原来的 1/8。

（3）作为无法直接通信的设备间的中转站。如果在两个设备无法直接通信的设备之间传送数据，可以在内存中开辟缓冲区进行中转，如将从卡片输入机读取的数据存到磁盘上，可以在内存中开辟一个缓冲区，先将卡片内容输入到缓冲区，然后再将缓冲区中的内容传送到磁盘上。

（4）解决程序所请求的逻辑记录大小和设备的物理记录大小不匹配的问题。例如，当进程需要从磁盘读取数据时，因为磁盘是块设备，以数据块作为数据传输单位，一次必须传输整个磁盘块，而程序仅要求读一条逻辑记录，此逻辑记录包含在一个盘块内，这时必须在内存中开辟一个缓冲区，其大小等于盘块大小，先将包含逻辑记录的盘块读到缓冲区中，然后进程再从缓冲区中提取所需的逻辑记录。

（5）加快进程（作业）的推进速度。假设某进程的功能是每次读取一张卡片，然后对此卡片上的信息进行处理，循环进行。如果在卡片阅读机和进程存储区之间直接传输数据，则在输入时，进程只能等待，当一张卡片输入完毕后才能进行处理，然后输入下一张卡片，再进行处理，依次类推。如果在卡片阅读机和进程存储区之间设置一个缓冲区，则输入时先将卡片信息输入到缓冲区中，然后将其复制到进程存储区，由进程对其进行处理，同时输入下一张卡片，如果两者速度匹配，则可使进程流畅运行而不必等待，从而加快了进程的推进速度。

在需要采取缓冲技术的场合，可以采用以下几种不同的缓冲技术：单缓冲、双缓冲、循环缓冲、缓冲池。

6.4.2 单缓冲

在单缓冲方式中，当用户进程发出 I/O 请求时，操作系统在内存中为其分配一个缓冲区。数据输入的过程是，当一个用户进程要求输入数据时，操作系统控制输入设备将数据送往缓冲区存放，再送往用户进程的数据存储区。

单缓冲方式由于只有一个缓冲区，这一缓冲区在某个时刻能存放输入数据或输出数据，但不能既是输入数据又是输出数据，否则缓冲区中的数据会引起混乱。在单缓冲方式下解决输入及输出的情形是，当数据输入到缓冲区时，输入设备工作，而输出设备空闲；当数据从缓冲区输出时，输出设备工作，输入设备空闲。因此单缓冲方式不能使外部设备并行工作，所以引进了双缓冲技术。

6.4.3 双缓冲

为输入或输出操作设置两个缓冲区 buffer1 和 buffer2。当进程要求输入数据时，输入设备将数据送往缓冲区 buffer1，然后进程从 buffer1 中取出数据进行处理；在进程从 buffer1

中取数据的同时，输入设备可向缓冲区 buffer2 中送入数据。当 buffer1 中的数据取完时，进程又可不需等待地从 buffer2 中提取数据，同时输入设备又可将数据送往 buffer1，如此交替使用 buffer1 和 buffer2 两个缓冲区。进程进行输出时的操作与输入类似。

双缓冲方式可以实现外部设备并行工作。先将数据输入到缓冲区 buffer1，然后进程从 buffer1 中取出数据进行计算，将输出的数据送往 buffer2，然后将 buffer2 中的数据送往输出设备，与此同时，从输入设备将数据输入到 buffer1，如此继续进行，使得输入设备和输出设备并行工作。

6.4.4　循环缓冲

循环缓冲技术是在内存中分配大小相等的存储区作为缓冲区，并将这些缓冲区连接起来，每个缓冲区中有一个指向下一个缓冲区的指针，最后一个缓冲区的指针指向第一个缓冲区。

使用循环缓冲结构需要两个指针，IN 指针指示可输入数据的第一个空缓冲区，OUT 指针指示可提取数据的第一个满缓冲区。系统初启时，这两个指针被初始化为 IN=OUT，在系统工作过程中，两个指针向同一个方向移动。输入时，数据输入到 IN 指针指示的缓冲区，输入完毕，IN 指针后移指向下一个可用的空缓冲区；当进程从循环缓冲结构提取数据时，提取 OUT 指针指示的缓冲区中的内容，提取完毕，OUT 指针后移指向下一个满缓冲区。指针移动时要检测空 / 满缓冲区的数量，当空缓冲区数量为 0 时，IN 指针不能再后移，此时 IN=OUT；满缓冲区数量为 0 时，OUT 指针不能再后移，此时 OUT=IN。

6.4.5　缓冲池

上述缓冲机制针对某特定的 I/O 进程和计算进程，是专用缓冲结构，而不是针对整个系统的公用缓冲结构。当系统较大时，会有很多这样的缓冲结构，这不仅要消耗大量的内存空间，而且利用率不高。为了提高缓冲区的利用率，目前普遍采用公用缓冲池结构，缓冲池中设置了可供多个并发进程共享的缓冲区。

1. 缓冲池的构成

对于既可用于输入又可用于输出的公用缓冲池，包含以下 3 种类型的缓冲区：空闲缓冲区、装满输入数据的缓冲区、装满输出数据的缓冲区。为了方便管理，将相同类型的缓冲区组成一个队列，于是可分别构成 3 个队列：空缓冲区队列 emq；装满输入数据的缓冲区队列 inq；装满输出数据的缓冲区队列 outq。这些缓冲区是公用的，因此要由操作系统进行统一管理。

2. 缓冲池的工作方式

公用缓冲池中的缓冲区有 4 种工作方式：收容输入、提取输入、收容输出、提取输出。

（1）收容输入。当进程需要输入数据时，从空闲缓冲区队列 emq 中获取一个空缓冲区，把它作为收容工作缓冲区，然后将数据输入其中，输入完毕，将它挂在 inq 队列的末尾。

（2）提取输入。当计算进程需要输入数据进行计算时，系统从 inq 队列的队首取得一

个缓冲区，作为提取输入工作缓冲区，计算进程从中提取数据，当进程提取完该缓冲区中的数据后，再将它挂在空缓冲区队列 emq 的末尾。

（3）收容输出。当计算进程需要输出数据时，系统从空缓冲队列 emq 的队首取得一个空闲缓冲区，将它作为收容输出工作缓冲区，装满数据后，将它挂在 outq 队列的末尾。

（4）提取输出。当要进行输出操作时，从输出缓冲区队列 outq 的队首取得一个缓冲区，作为提取输出工作缓冲区，当数据输出完毕，将该缓冲区挂在空缓冲区队列 emq 的末尾。

6.5 设备分配

在多任务操作系统中，并发进程共享系统中的设备。为了合理利用系统设备，达到一定的系统目标，不允许进程自行决定设备的使用，而是由系统按一定原则统一分配、管理。进程要进行 I/O 操作时，需向操作系统提出 I/O 请求，然后由操作系统根据系统当前的设备使用状况，按照一定的策略，决定对该进程的设备分配。

6.5.1 与设备分配相关的因素

1. I/O 设备的固有属性

从设备分配的角度看，I/O 设备可以分为独占设备和共享设备两种，有些设备仅适应于某进程独占，有些设备可以被多进程共享。

独占设备被分配给一个进程后，就被该进程独占使用，其他任何进程不能使用，直到该进程使用完毕（不再使用）主动释放为止。在进程占有独占设备期间，设备不一定总在工作，但是，即使设备空闲时，也不能被其他进程使用。例如打印机，如果两个进程交替使用，就会使得两个进程打印的内容混淆在一起无法辨别，所以必须独占使用打印机，以使得打印内容完整。独占设备是一种临界资源，对独占设备的竞争可能引起死锁。

共享设备是指可以由多个进程交替使用的设备，如磁盘机。

对独占设备通常采用静态分配方式。在一个作业开始执行前进行独占设备的分配，一旦把某独占设备分配给作业，就被该作业独占（永久地分配给该作业），直到作业结束撤离时，才由操作系统将分配给作业的独占设备收回。静态分配方式实现简单，但是设备利用率不高。

对共享设备通常采用动态分配方式，即在作业（进程）运行过程中，当进程需要使用设备时，通过系统调用命令向系统提出 I/O 请求，系统按一定策略为进程分配所需设备，进程一旦使用完毕就立即释放该设备。共享设备一旦完成当前 I/O 工作就被释放，从而使多个并发进程可以交替使用此设备，设备利用率高。

2. I/O 设备的分配算法

当多个并发进程对同一设备提出 I/O 请求时，要采用一定策略来决定将此设备分配给哪个进程，这就是分配算法的功能。通常有两种设备分配算法。

（1）先来先服务算法。当进程提出 I/O 请求时，要先建立相应的数据结构——I/O 请求块，然后按一定原则将 I/O 请求块放入相应设备的设备请求队列中。

在先来先服务算法中，对同一设备，按并发进程中 I/O 请求发出的时间先后，将 I/O 请求块排队。当设备可用（可以分配）时，将设备分配给队首元素对应的进程，即最先对此设备提出 I/O 请求的进程。

（2）优先级算法。按一定原则设置进程的优先级，按进程的优先级由高到低对各并发进程对同一设备的 I/O 请求块排队。当设备可用（可以分配）时，将设备分配给队首元素对应的进程——在对同一设备提出 I/O 请求的并发进程中优先级最高的进程。

优先级算法的依据是，在进程调度中优先级高的进程优先获得处理机，若对它的 I/O 请求也赋予较高的优先级，则有助于该进程尽快完成，从而尽早释放它所占用的资源，以防系统发生死锁，并提高资源利用率。如果系统进程自身也希望使用 I/O 设备而提出 I/O 请求时，它应比用户进程的 I/O 请求具有更高的优先级，对于优先级相同的 I/O 请求，应按时间先后顺序排队。

3. 设备分配的安全性

多个并发进程对同一组资源的竞争可能引起死锁。设备分配与死锁密切相关，因此进行设备分配时要考虑到设备分配对系统安全性的影响。

对独占设备的静态分配不会引起死锁，因为是在作业开始执行前对独占设备进行分配，只要该作业需要使用该设备且该设备可分配，就将该独占设备永久分配给该作业，直至作业结束时由系统收回。若作业分配不到其所需的独占设备，作业不能运行，从而也不分配其他所需设备，这样不符合死锁的“请求且保持”条件。

对共享设备进行动态分配时，从系统安全性角度看，有如下两种分配方式。

（1）安全分配方式。在这种分配方式中，当进程通过系统调用发出 I/O 请求后，进程立即进入阻塞状态，直到所提出的 I/O 请求完成才被唤醒。因为进程阻塞时不会继续请求其他设备，进程运行时已经释放曾经占有的、已经完成 I/O 操作的共享设备，所以一个进程在任何时刻都不可能在占有一个共享设备的同时提出对其他共享设备的请求，因此不符合死锁的必要条件——占有且申请条件，不会发生死锁。这种分配方式使得进程进展速度慢。

（2）不安全分配方式。在这种分配方式中，进程发出 I/O 请求后继续运行，在运行过程中可以发出第二个 I/O 请求、第三个 I/O 请求、……。仅当进程所请求的设备已经被另一个进程占用时，请求进程才进入阻塞状态。

这种分配方式使一个进程可以同时操作多个设备，使进程进展迅速。但是可能具备“占有且申请条件”，可能会发生死锁。

4. 设备无关性

设备无关性是指当在应用程序中使用某类设备时，不直接指定具体使用哪个设备，而只指定使用哪类设备，由操作系统为进程分配具体的一个该类设备。

当应用程序（进程）具体指定使用哪个设备时，如果该设备已经分配给其他进程或出了故障，而此时可能还有几台其他的同类设备可用，但该进程却被阻塞。若进程只指定使用哪类设备，则可从该类设备中任选一个分配给它使用，只有当所有此类设备全部分配完毕时，进程才会阻塞。

设备无关性功能可以使应用程序的运行不依赖于特定设备是否完好、是否空闲，而由系统合理地进行分配，从而保证程序的顺利运行。

为了便于描述设备无关性，引入逻辑设备和物理设备这两个概念。逻辑设备指示一类设备，物理设备指示一台具体的设备。在应用程序中使用逻辑设备名请求使用某类设备，在实际进行 I/O 操作时，在物理设备上进行，因此，系统必须具有将逻辑设备名转换为物理设备名的功能。这类似于存储管理中逻辑地址和物理地址的概念，应用程序中使用的是逻辑地址，而在实际执行过程中必须使用物理地址，存储管理要具有地址变换的功能。

6.5.2 虚拟设备技术

系统中独占设备的数量有限，且对独占设备的分配往往采用静态分配方式，这样做不利于提高系统效率。这些设备只能分配给一个作业，且在作业的整个运行期间一直被占用，直至作业结束时才被释放。但是，一个作业往往不能充分利用该设备，在独占设备被某个作业占用期间，往往只有一部分时间在工作，其余时间处于不工作的空闲状态，因此设备利用率低，其他申请该设备的作业因得不到该设备而无法运行，降低了系统效率。

另一方面，独占设备往往是低速设备，因此，在作业执行过程中，由于要等待这类设备传输而大大延长了作业的执行时间。

为了克服独占设备的这些缺点，可以采用虚拟设备技术，即用一共享设备——高速存储设备（如高速磁盘机）上的存储区域模拟独占设备。虚拟设备技术的关键是预输入、缓输出。

（1）预输入：在作业执行前，操作系统先将作业信息从独占设备预先输入到高速外存中。此后，作业执行中不再占有独占输入设备，使用数据时不必再从独占设备输入，而是从高速外存中读取。

（2）缓输出：在作业执行过程中，当要进行输出操作时，不必直接启动独占设备输出数据，可以先将作业输出数据写入高速外存中，在作业执行完毕后，再由操作系统来组织信息输出。

由于作业运行期间不占有独占设备，从而使一台独占设备可以为多个作业共享，提高了独占设备的利用率；而且在作业执行过程中不直接和慢速的独占设备打交道，因此缩短了作业的执行时间。

6.6 SPOOLING 系统

6.6.1 SPOOLING 系统的简介

在早期，大型主机的吞吐量很重要。因为，当时计算机资源短缺、价格昂贵，人们希望能在一台主机上做尽量多的工作，为更多的用户服务。

为了提高大型主机的吞吐量，采用了脱机输入输出技术。如果直接在大型主机上进行联机输入输出，则慢速输入输出操作占用机器运行时间的比例很大，系统吞吐量很低。因

此，额外添置专门进行输入输出的配置简单的卫星机。输入作业时，用一台专用的卫星机把作业输入到高速磁盘中，然后把高速磁盘连接到大型主机上，主机高速读取磁盘上的信息。作业输出时，由主机把输出结果写入高速磁盘，然后将高速磁盘连接到另一台专用的卫星机上，在卫星机的控制下，在低速输出设备上输出作业结果。

脱机输入输出需要人工干预，产生人工错误的机会多，且效率低。当硬件发展出现了通道和中断技术，软件发展出现了多道程序设计技术后，采用这些技术和虚拟设备技术，产生了 SPOOLING 技术来取代脱机输入输出技术。SPOOLING（Simultaneaus Peripheral Operating On Line）的含义是外围设备同时联机操作，又称为假脱机操作。它不需要额外配置卫星机，而是将输入输出设备同时连接在主机上，是对脱机输入输出的模拟。用一道输入程序模拟脱机输入时的卫星机，通过通道把低速 I/O 设备上的数据传送到高速磁盘上；用另一道程序模拟脱机输出时的卫星机，通过通道把数据从磁盘传送到低速输出设备上。

6.6.2　SPOOLING 系统的组成

1. 输入井和输出井

输入井和输出井是在高速磁盘上开辟的两个存储区域。此处采用了虚拟设备技术，用输入井虚拟低速输入设备，用输出井虚拟低速输出设备。输入井用于暂时存储从输入设备预输入的信息，输出井用于暂时存储要缓输出到输出设备的信息。利用虚拟设备技术，可以提高作业（进程）的执行速度，提高 I/O 设备的利用率。SPOOLING 系统的构成如图 6-8 所示。

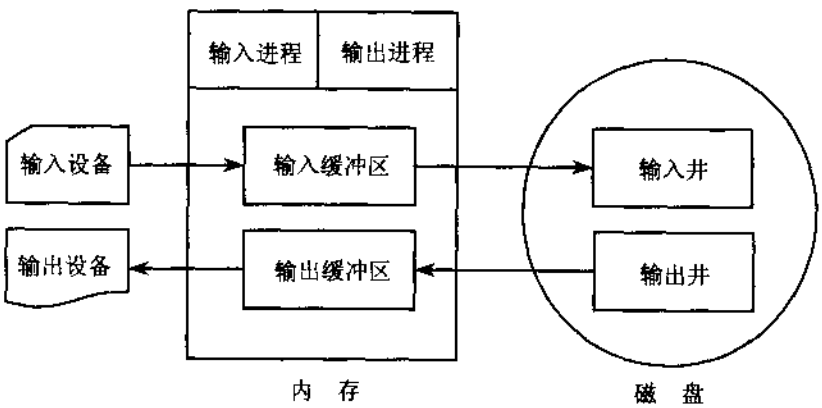


图 6-8　SPOOLING 系统的构成

2. 输入缓冲区和输出缓冲区

I/O 输入设备无法直接和磁盘存储设备进行数据传输，必须经过内存的中转，因此对应于输入和输出，要在内存中分别开辟输入缓冲区和输出缓冲区。

输入缓冲区用作输入设备和磁盘输入井之间的中转站，输出缓冲区用作磁盘输出井和输出设备之间的中转站。

3. 输入进程和输出进程

输入进程模拟脱机输入时的卫星输入机，它将数据从输入设备经过输入缓冲区送到输

入井中。当 CPU 需要读取数据时，直接从输入井中提取数据到内存。

输出进程模拟脱机输出时的卫星输出机，它将应用进程要输出的数据送到输出井，当输出设备空闲时，将输出井中的数据经过输出缓冲区送到输出设备输出。

6.7 I/O 控制过程

当用户进程通过系统调用提出 I/O 请求时，从系统响应此请求开始，到系统完成用户要求的 I/O 操作，并唤醒相应的等待 I/O 完成的进程为止，整个过程称为 I/O 控制过程。

I/O 控制过程包括以下步骤：

（1）响应 I/O 请求，为在具体物理设备上进行 I/O 操作做准备，包括将逻辑设备名转换为物理设备名（设备分配）、I/O 请求的合法性等。

（2）设备驱动，控制设备完成 I/O 操作，对每类设备分别设置不同的设备驱动程序。

（3）中断处理，I/O 操作完成后，设备控制器向 CPU 发送中断信号，CPU 响应后转向相应的中断处理程序进行善后处理。

任何操作系统的 I/O 控制过程都由以上 3 个步骤构成，其中设备驱动是核心操作步骤。不同的操作系统可能有不同的进入设备驱动的方式，如下所述：

（1）为每一类设备设置一个进程（进程是程序的运行过程，不是静态的程序），专门用于响应对这类设备的 I/O 请求，控制这类设备的 I/O 操作。例如，为同一类型的打印机设置一个打印进程，专门处理对此类型打印机的 I/O 请求。

（2）在整个系统中设置一个 I/O 进程，专门处理对所有各类设备的 I/O 请求，然后转到相应的设备驱动程序。

（3）不设置专门的 I/O 控制进程，而由进程自己调用相应的设备驱动程序。

在设置 I/O 进程的系统中，I/O 进程循环执行检测 I/O 请求队列，并处理 I/O 请求，在 I/O 请求队列为空时处于睡眠状态，当有新的 I/O 请求到来时被唤醒。

6.7.1 用户进程的 I/O 请求

用户进程通过系统调用命令提出 I/O 请求，用户编写源程序时不一定使用到系统调用命令，且不一定意识到 I/O 操作是通过系统调用完成的，而是利用了编译系统提供的库函数。例如，编写 C 程序时，可以利用 C 编译系统提供的输入输出函数来提出 I/O 请求，如 printf()、scanf()函数。这些函数不是系统调用，而是通过系统调用实现的，这些库函数中包含系统调用命令。

用户进程的 I/O 请求包括进行 I/O 操作的逻辑设备名、要求的操作、传送数据在内存中的起始地址、传送数据的长度。I/O 请求中要求的操作是抽象的、高级的，如打开、关闭、读、写设备等，每一个抽象操作在具体物理设备上要经过若干步基本操作才能完成。

6.7.2 设备驱动

设备驱动程序的任务是控制具体的物理设备完成 I/O 请求中的抽象操作，其任务是通

过向设备控制器发送一系列命令字完成的。

设备驱动程序与设备控制器和 I/O 设备的硬件特性密切相关，而计算机系统中 I/O 设备种类繁多、特性各异、操作方法不同，因此要首先从设备驱动角度对设备分类，为每一类设备配置专用的设备驱动程序。有时即使同一类型的设备，由于生产厂家不同，也可能不完全兼容，必须为它们配置不同的驱动程序。

设备驱动程序与 I/O 设备所采用的 I/O 控制方式密切相关。常用的 I/O 控制方式是中断方式和 DMA 方式，这两种方式的设备驱动程序显然不同，后者应按数组方式启动设备及进行中断处理。

6.8　磁盘 I/O

磁盘相对于其他外部存储设备容量大、存取速度快、可以随机存取，因此是当前存放程序和数据最理想的外部存储设备，程序和数据以文件的方式存放在磁盘上。计算机运行过程中，要经常访问文件，因此磁盘 I/O 速度快慢直接影响到系统性能。改善磁盘 I/O 性能是当代操作系统的一个重要任务。

6.8.1　磁盘存储格式

磁盘可包含一个或多个圆形盘片，每片有两面。每面划分为若干半径不同的同心磁道，然后将每个磁道等分为若干扇区。各磁道虽然周长不同，但存储容量相同，各扇区的存储容量也相同，一般每个扇区的容量为 512 字节，内层磁道的存储密度比外层磁道高。

磁盘存储空间分配以整数 N 个扇区为单位，称为簇，不同的操作系统，N 可有不同的取值。一个文件在磁盘上占有整数个簇，一个簇只能分配给一个文件使用。磁盘 I/O 以扇区为单位，一次传送一个扇区。

6.8.2　磁盘 I/O 性能

一般以磁盘存取速度来衡量磁盘 I/O 性能。对磁盘的存取操作包括 3 个步骤：首先磁头径向运动到指定磁道；然后旋转盘片，将指定要访问的扇区旋转到磁头下，使磁头处于扇区开始位置；最后开始读或写数据。因此磁盘访问时间可以分为以下 3 部分。

（1）寻道时间 T_s。把磁头移到指定磁道上用的时间。

（2）旋转延迟 T_r。指定扇区旋转到磁头下经历的时间。

（3）传输时间 T_t。把数据从磁盘读出或向磁盘写入数据所经历的时间，由读写的字节数和磁盘旋转速度决定。

$$T_t = \frac{b}{rN}$$

其中，b 为一次读写的字节数，r 为磁盘每秒的转数，N 为每条磁道的字节数。

对磁盘的访问时间为三部分时间之和 $T=T_s+T_r+T_t$。在这三个时间中，寻道时间和旋转延迟与读写的字节数无关，寻道时间所占比例最大。

6.8.3 磁盘调度

磁盘可供多个并发进程共享，当有多个并发进程同时要求访问磁盘时，应该考虑采取一种调度策略，使各进程对磁盘的平均访问时间最小。由于在访问磁盘的时间中，主要是寻道时间，因此，磁盘调度的目标是使磁盘的平均寻道时间最短。常用的磁盘调度算法有先来先服务算法、最短寻道时间优先算法和扫描算法。

1. *先来先服务算法*

这种调度算法按进程请求访问磁盘的时间先后次序进行调度。此算法的优点是实现简单且公平，每个进程的磁盘 I/O 请求都能依次得到处理，不会出现某一进程的请求长时间得不到满足的情况。缺点是未对寻道进行优化，平均寻道时间较长，如表 6-1 所示。

2. *最短寻道时间优先算法*

该算法选择磁盘 I/O 请求的原则是其要访问的磁道与当前磁头所在的磁道距离最近，以使每次的寻道时间最短。此算法只从当前角度考虑，没有考虑全局，表面看来平均寻道时间应该最短，但是不一定。

最短寻道时间优先算法可能导致某个进程发生“饥饿”现象。因为，只要不断有新进程的请求到达，且其所要访问的磁道与磁头当前所在磁道的距离最近，这种新进程的 I/O 请求必须优先满足，这样就会使某些进程的磁盘 I/O 请求长时间得不到满足，如表 6-2 所示。

表 6-1 先来先服务算法

（从 100 号磁道开始）	
选择访问的下一个磁道号	磁头移动距离（磁道数）
55	45
58	3
39	19
18	21
90	72
160	70
150	10
38	112
184	146
平均寻道长度：498/9=55.3	

表 6-2 最短寻道时间优先算法

（从 100 号磁道开始）	
选择访问的下一个磁道号	磁头移动距离（磁道数）
90	10
58	32
55	3
39	16
38	1
18	20
150	132
160	10
184	24
平均寻道长度：248/9=27.5	

3. *扫描算法*

为防止出现最短寻道时间优先调度算法的“饥饿”现象，对最短寻道时间优先调度算法进行了改进，提出了扫描算法，如表 6-3 所示。

扫描算法不仅考虑到要访问的磁道与当前磁道间的距离，更优先考虑的是磁头当前的移动方向。例如，当磁头正从里向外移动时，扫描算法选择的下一个访问对象是在当前磁头所在磁道之外距离当前磁道最近的磁道，这样从里向外地访问，直至再无更外层的磁道需要访问时，磁头才返回，从外向里移动，同样每次也是选择当前磁道中距离最近的磁道，到头后再返回，从里向外移动访问。由于此算法中磁头移动的规律很像电梯的运行，因此又称为电梯调度算法。

4. 循环扫描算法

扫描算法能获得较好的寻道性能，且能防止“饥饿”现象，所以被广泛应用于大、中、小型计算机和网络中的磁盘调度。但它存在一个问题：当磁头刚从里向外移动而越过了某一磁道时，恰好又有一个进程请求访问此磁道，这时，该进程必须等待，待磁头继续从里向外，然后再从外向里扫描完所有要访问的磁道后，才处理该进程的请求，致使该进程的请求被大大推迟。为了减小这种延迟，引入循环扫描算法，规定磁头单向移动。例如，只从里向外移动，当磁头移到最外的磁道并访问后，磁头立即返回到最里的要访问的磁道，再从里向外移动，如表 6-4 所示。

表 6-3　扫描算法

（从 100 号磁道开始）	
选择访问的下一个磁道号	磁头移动距离（磁道数）
150	50
160	10
184	24
90	94
58	32
55	3
39	16
38	1
18	20
平均寻道长度：250/9=27.8	

表 6-4　循环扫描算法

（从 100 号磁道开始）	
选择访问的下一个磁道号	磁头移动距离（磁道数）
150	50
160	10
184	24
18	166
38	20
39	1
55	16
58	3
90	32
平均寻道长度：322/9=35.8	

6.8.4　磁盘高速缓存

当进程从磁盘读取数据时，为了提高读盘速度，可以采用缓冲技术，即在内存中开辟一个缓冲区用于接收从磁盘读取的数据，这个缓冲区称为磁盘高速缓存，缓冲区的大小与磁盘块相匹配，其工作原理类似于内存和 CPU 之间的高速缓存。

当有一个进程请求访问某盘块中的数据时，先查看磁盘高速缓存，看其中是否有进程所需访问的盘块数据的备份。若有，则直接从磁盘高速缓存中提取数据，而不必访问磁盘；

若没有，则先将所需盘块读到磁盘高速缓存，然后从中提取数据，送往请求进程的数据存储区。

6.9 Linux 系统的设备管理

6.9.1 Linux 系统设备管理的特点

Linux 系统把设备分为两类。

（1）块设备。用于存储信息，它对信息的存取是以信息块为单位的，如通常使用的磁盘、磁带等。

（2）字符设备。通常用于输入输出，作为人和计算机之间的接口，它对信息的存取是以字符为单位进行的，如键盘、鼠标、显示器、打印机等。

Linux 系统把 I/O 设备看作文件，称为特别文件。例如，打印机的文件名为 lp，控制台终端的文件名是 console。这些特别文件组织在目录/dev 下。如果要访问打印机可以使用路径名/dev/lp。在高级编程语言中，可以使用以下语句打开特别文件/dev/lp。

fd=open("/dev/lp",O-WRONLY);

fd 为打开文件号，/dev/lp 为打开文件名，O-WRONLY 为打开方式。

Linux 的这一特征使得任何外部设备在用户面前与普通文件完全一样，而不必涉及它的物理特性，给用户带来极大的方便。在文件系统内部，外部设备和普通文件一样受到存取控制的保护，仅仅在最终驱动设备时才转向各个设备的驱动程序。

块设备上的普通文件和目录文件并不是设备特殊文件，但块设备本身仍可作为特殊文件来访问。

6.9.2 Linux 设备驱动程序的接口

1. 文件系统与设备驱动程序的接口

Linux 系统为各类设备分别配置不同的驱动程序，在用户程序中通过文件操作方式使用设备，如打开、关闭、读、写，由文件系统转入设备驱动程序。

核心与设备驱动程序的接口是块设备开关表和字符设备开关表。每一种设备类型在表中占用一个表目，包含若干数据项，其中有一项为该类设备驱动程序入口地址，在系统调用时引导核心转向适当的驱动程序接口。

对设备特殊文件的系统调用，根据文件类型转入块设备开关表或字符设备开关表进行打开、关闭块设备或字符设备的操作。字符设备特殊文件的系统调用 read、write 转向字符设备开关表中指示的设备驱动程序，而对普通文件或目录文件的 read、write 系统调用则通过高速缓冲模块转向设备驱动模块中的策略（Strategy）过程。

如图 6-9 所示为文件系统与设备驱动程序的接口结构。

2. 设备开关表

Linux 系统把块设备和字符设备又分别细分为若干类。如块设备可分为硬盘、软盘、

磁带、光盘等，字符设备可分为终端设备、打印机等。为指定一类设备，将设备类从 0 开始顺序编号，称为主设备号；同一类的设备可能有许多，为了指定一台具体设备，需要一个次设备号来标识。因此，在指定一台具体设备时要给出块设备/字符设备、主设备号、次设备号，如表 6-5 所示。

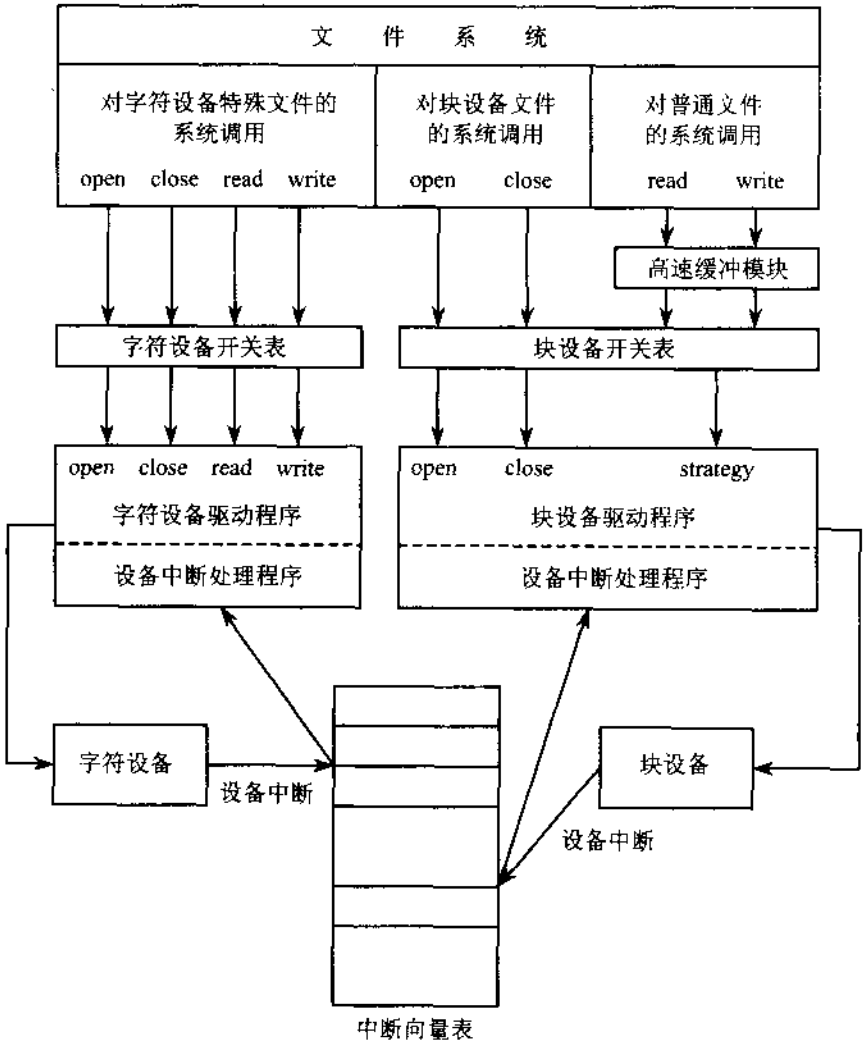


图 6-9　文件系统与设备驱动程序的接口

表 6-5　块设备开关表

操作 / 主设备号	open	close	strategy（read、write）
0			
1			

设备开关表相当于一个二维矩阵，每一行含有同一类设备的驱动程序入口地址，主设备号与行号一一对应；每一列是完成不同操作（open、close、read、write）的驱动程序的入口地址，如表 6-6 所示。

表 6-6 字符设备开关表

操作 / 主设备号	open	close	read	write
0				
1				

6.9.3 Linux 的磁盘高速缓存

1. 磁盘高速缓存

对文件系统的一切存取操作都能通过直接从磁盘上读或写来实现，但磁盘 I/O 的速度较慢，系统性能较低。为了减少对磁盘的存取频率，Linux 使用了磁盘高速缓存技术。

Linux 磁盘缓冲管理策略试图把尽可能多的有用数据保存在缓冲区中。磁盘缓冲管理模块位于文件系统和块设备驱动程序之间。

当从磁盘中读数据时，文件系统先从磁盘高速缓存中读，如果数据已在高速缓存中，则不必启动磁盘 I/O，如果数据不在高速缓存中，则启动磁盘 I/O，从磁盘读取数据送往高速缓存，进程再从高速缓存中读取数据。

当进程往磁盘上写数据时，先往高速缓存中写，以便随后读它时能从高速缓存中读取，而不必启动磁盘读取。Linux 采取了“延迟写”策略，即如果缓冲区还没有写满，则不急于把缓冲区的内容写到磁盘上，而是在缓冲管理数据结构中对该缓冲区设置延迟写标志，当高速缓存中的数据延迟到必须往磁盘上写的时候才进行写盘操作。

2. 磁盘缓冲管理的数据结构

Linux 使用多个内存缓冲区进行磁盘缓冲，为了合理有效地使用这些缓冲区，必须按一定策略进行管理。管理的依据是缓冲区的相关信息，为了能访问这些信息，必须将其按一定的数据结构进行组织。

首先，给每个缓冲区建立“缓冲首部”来存放该缓冲区的相关信息（见表 6-7）。其内容有：

（1）设备号 dev：缓冲区内包含的信息所属设备的设备号。

（2）块号 blkno：该缓冲区对应的磁盘的物理块号。

（3）状态 flag：描述了缓冲区的当前状态，包括：

- 忙标志位 BUSY：缓冲区当前是否正忙。
- 有效位 AVE：缓冲区包含的数据是否有效。
- 延迟写位 DELWR：是否延迟写。
- 写标志位 WRITE：是否正在把缓冲区的内容写到磁盘上。
- 读标志位 READ：是否从磁盘往缓冲区读取信息。

表 6-7　缓冲首部

设备号 dev	b-back
块号 blkno	av-forw
状态 flag	av-back
b-forw	

- 等待位 WAIT：是否有一个进程正在等待该缓冲区。
- 设备缓冲区队列前向指针 b-forw。
- 设备缓冲区队列后向指针 b-back。
- 空闲缓冲区队列前向指针 av-forw。
- 空闲缓冲区队列后向指针 av-back。

要对所有缓冲区进行管理，必须将所有缓冲首部组织成一定的数据结构，Linux 建立了两个缓冲区队列：空闲缓冲区队列和设备缓冲区队列。空闲缓冲区队列如图 6-10 所示。

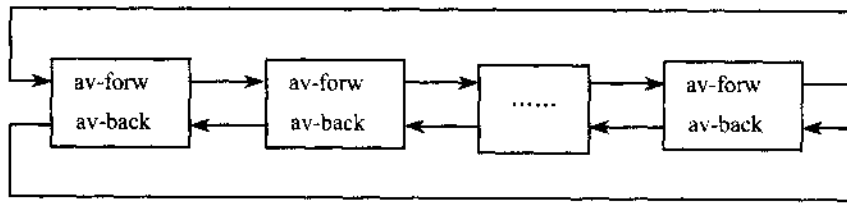


图 6-10　空闲缓冲区队列

为了对缓冲区进行分配，把所有空闲缓冲区组织成一个队列，此队列中的所有缓冲区的忙标志位 BUSY 均为 0。该队列是一个双向循环链表。

对于每类设备都建立一个设备缓冲区队列，由与该类设备有关的所有缓冲区构成。此队列中的所有缓冲区首部的忙标志位 BUSY 均为 1。该队列是一个双向循环链表。

任何缓冲区首部要么位于空闲缓冲区队列中，要么位于设备缓冲区队列中。设备缓冲区队列如图 6-11 所示。

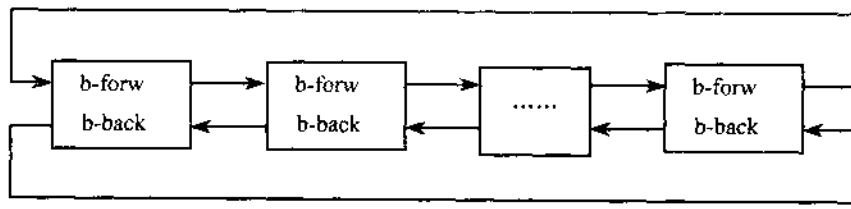


图 6-11　设备缓冲区队列

3. Linux 磁盘缓冲区管理算法

Linux 提供的磁盘高速缓冲区不属于某进程专用，而是由多进程共享，由操作系统进行统一管理。为了提高使用效率，必须采取适当的管理策略。

当某进程要从一个磁盘读取数据时，先检查要读取的磁盘块是否包含在某缓冲区中，如果不在，则从空闲缓冲区队列中分配给它一个空闲缓冲区；当某进程要将数据写入磁盘时，先看一下要写的磁盘块是否在某缓冲区中，如果不在则为这个磁盘块分配一个空闲缓冲区，被分配的空闲缓冲区从空闲缓冲区队列中退出，进入相应设备的设备缓冲区队列，该缓冲区的 BUSY 位置 1。

当缓冲区的信息读到相应进程的内存区后，或进程信息写到缓冲区后，使释放进程使用的缓冲区，将 flag 中的 BUSY 位置 0，送入空闲缓冲区队尾，即将置为延迟写的缓冲区也送入空闲缓冲区队列。这样可以使有限的缓冲区得到充分利用，满足多个并发进程的需求。

当一个缓冲区被送往空闲缓冲区队尾时，并不退出设备缓冲区队列，而仍然留在原队列中。这样做的原因是，此次使用完毕送往空闲缓冲区队列的缓冲区，其保存的信息可能被再次访问。如果将其保留在设备缓冲区队列中，当后来要读取的磁盘数据仍然保留在此缓冲区中时，可以直接从此缓冲区读，而不必启动磁盘 I/O，提高了读盘速度，这正是使用磁盘高速缓存的目的。

如果要将一个使用完毕送入空闲缓冲区队列的缓冲区重新分配给其他设备，则可以将该缓冲区同时从空闲缓冲区队列和原设备缓冲区队列中抽出，送入新的设备缓冲区队列。

因为被使用过的缓冲区放在空闲缓冲区队列的末尾，随着不断地从空闲缓冲区队列上摘下缓冲区，后面的缓冲区会不断向队首移动。如果一个延迟写标志位为 1 的缓冲区移到空闲缓冲区队列的头部，当其被重新分配之前，要先把内容写到相应设备的指定磁盘块中。

6.9.4 Linux 的字符设备缓冲区管理

1. 字符缓冲区及其队列

Linux 中为字符设备的数据传输设置了一个公用的字符缓冲池，该缓冲池内含有若干个缓冲区，但每个缓冲区很小，只含几个或几十个字节。为合理使用这些缓冲区，将其组织成一定的数据结构。同块设备缓冲区的管理类似，建立空闲字符缓冲区队列和设备字符缓冲区队列。每个字符缓冲区由 4 部分构成：该缓冲区第一个字符的位置、最后一个字符的位置、指向下一个字符缓冲区的指针、字符缓存区。

字符缓冲区如图 6-12 所示。

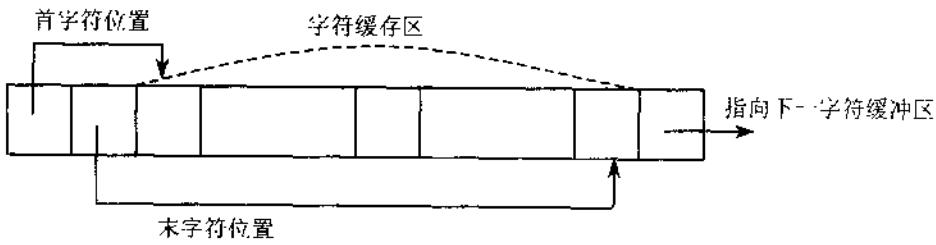


图 6-12 字符缓冲区

对字符缓存区的操作是每次送入一个字符或取出一个字符。送入字符在字符缓存区的

尾部进行，即放在最后一个字符的后面；取出字符从字符缓存区的首部进行，即取第一个字符。最后一个字符的位置不一定是字符缓存区的末尾，随着不断往字符缓存区中放入字符，最后一个字符的位置会向后推移，直至字符缓存区的末尾；第一个字符的位置不一定是字符缓存区的开始位置，随着不断地从缓存区中取出字符，第一个字符的位置不断后移直至字符缓存区的末尾。

2. 空闲字符缓冲区的分配和回收

在一进程用字符设备进行 I/O 操作时，文件系统为该字符设备分配一个空闲缓冲区。分配时要检测空闲缓冲区队列，当队列非空时，从队首取下一个空闲缓冲区分配给该字符设备，送入该字符设备的字符缓冲区队列，并把指向该缓冲区的指针返回给调用进程；当队列空时，表明已经没有缓冲区可分配，则请求 I/O 操作的进程被阻塞。

当字符缓冲区中的数据被提取完后，便释放该缓冲区，将该缓冲区送入空闲字符缓冲区队列的首部。（空闲字符缓冲区队列其实是一个栈。）若此时有因申请空缓冲区而阻塞的进程，则将其唤醒。

6.9.5 Linux 的设备驱动

设备驱动是指控制具体的物理设备完成 I/O 操作，与硬件的物理特性和数据传输控制方式密切相关，不同类型的设备、不同数据传输控制方式要采用不同设备驱动方式。

Linux 首先把设备分为块设备、字符设备两大类，然后再对块设备和字符设备具体分类，针对每类设备，分别开发不同的设备驱动程序。

1. 块设备驱动

（1）数据结构。

1）块设备表。对每一类块设备，分别设置块设备表，记录该类设备的相关信息。其内容包括：

忙标志：标志设备的忙闲状态，0 表示空闲，1 表示正忙。

出错次数：指设备 I/O 出错次数。每次 I/O 完成后，就转入设备中断处理程序，设备中断处理程序检测此次 I/O 过程是否出错，如果出错，则将出错次数加 1，如果出错次数没有超过一个规定值，则中断处理程序重新启动一次数据传输，如果出错次数超过规定值，则认为是真正的数据传输错误，不再重新启动数据传输。

设备缓冲区队列头指针：分配给该设备的缓冲区队列的头指针。

设备缓冲区队列尾指针：分配给该设备的缓冲区队列的尾指针。

I/O 请求队列头指针：请求该类设备 I/O 操作的请求块组成的队列的头指针。

I/O 请求队列尾指针：请求该类设备 I/O 操作的请求块组成的队列的尾指针。

2）I/O 请求队列。用户进程的 I/O 请求包括要求完成 I/O 操作的逻辑设备名、要求的操作、传送数据在内存中的起始地址、传送数据的长度，将这些信息组织成 I/O 请求块 iorb。逻辑设备名相同（同一类设备）的 I/O 请求块构成一个队列，称为 I/O 请求队列。每类设备都分别有一个 I/O 请求队列。

（2）磁盘设备驱动程序。

磁盘分硬盘、软盘，在此仅以硬盘为例讨论块设备的驱动程序，其他块设备的驱动程序与硬盘类似。

驱动程序控制设备具体实现进程的高级 I/O 请求包括打开、关闭、读、写等。

1）打开。硬盘在使用之前必须进行打开操作，其目的是在使用硬盘之前进行必要的初始化——设置硬盘的工作参数。硬盘工作参数存放在主存中操作系统数据结构硬盘控制块中，它是驱动程序对硬盘进行控制的依据。硬盘工作参数存放在硬盘的引导区中，因此要将引导区内容读到硬盘控制块中。

为了读硬盘引导区，必须初启硬盘，首先测试该设备是否就绪（通过测试设备控制器中的设备状态寄存器的 ready 位判断），如果已就绪再测试该硬盘是否可用。设备不能使用的原因可能是没有格式化或其他一些硬件故障，测试的方法是读取该硬盘的首标，如果不能被正确读出，说明硬盘不能使用，否则是可用的。如果在限定时间内设备未就绪或经测试不能使用，则设备初启失败。若初启成功，便读取硬盘引导区，初始化硬盘控制块。

2）读写硬盘

由策略接口程序 strategy 完成，每次只读写一个扇区。Linux 中对硬盘有两种读方式和三种写方式。

一般读：把当前所需盘块中的内容读入缓冲区。

预先读：进程读顺序文件的盘块时，会预见到要读的下一个盘块，因此读出当前块后，可提前读下一盘块。这样若以后需要读该盘块，因为已经提前读入，所以不必启动磁盘操作，提高了进程的读盘速度。

一般写：把缓冲区数据写到盘块，且进程必须等待写操作完成。

异步写：把缓冲区数据写到盘块，但进程不必等待写操作完成。

延迟写：当进程要把数据写到盘上时，先写到缓冲区，然后将该缓冲区标记为延迟写，挂到空闲缓冲区队列的末尾，但并不立即将该缓冲区的内容写到磁盘。

硬盘读写驱动过程为：将逻辑块号转换为物理块号；申请 DMA；将物理块号转换为盘面号/柱面号/扇区号；移动磁头到指定柱面；向硬盘控制器发送读写命令。CPU 发出读写命令后不必等待，而继续执行其他进程。当传输结束后，硬盘控制器向 CPU 发送中断信号，CPU 响应后转入硬盘中断处理程序。

硬盘中断处理程序完成以下工作：检测此次数据传输是否出错，传输错误记录在硬盘控制器的状态寄存器中，如果出错，进行错误处理；由于磁盘读写一次只能传输一个扇区，每传输一个扇区就中断一次，而 I/O 读写请求缓冲区内可能有大于一个扇区的数据要传输，因此每次传输结束，中断处理程序要检查 I/O 读写请求缓冲区内是否还有数据要传输，若有，则继续传输余下的数据，若没有，说明一个缓冲区已经传输完毕。若 I/O 读写请求缓冲区内的数据全部传输完毕，则唤醒等待传输完成的进程。

2. 字符设备驱动

字符设备用作人和计算机之间的接口设备，人们通常通过字符设备向计算机输入信息（指令、数据）或输出信息。字符设备以字符作为数据传输单位，速度较慢。常见的字符设备有终端、行式打印机、卡片输入机、纸带输入机等。

字符设备的传送用一组专用的寄存器来实现。每种字符设备的控制器一般都有3个寄存器：控制寄存器接收CPU发送来的命令，控制设备的操作；状态寄存器保存设备的状态；数据寄存器暂存要传送的数据。

（1）数据结构。对每类字符设备分别建立字符设备表，记录使用该类字符设备所需的各种信息。由于字符设备种类繁多，不同种类的字符设备特性差别较大，因此其字符设备表差别也较大，在此不一一列举。

（2）终端驱动程序。字符设备种类繁多，其驱动程序各不相同，在这里以终端驱动程序为例来讲述Linux中字符设备驱动程序的工作原理。

终端主要由键盘和显示器构成，终端驱动程序控制终端设备和进程之间的字符数据传输，包括以下5个操作：

- ttopen：打开终端机，建立终端机和终端进程之间的对应关系。
- ttclose：切断终端机和终端进程的联系。
- ioctl：用于对终端机的控制，例如状态设置、测试、终端机属性值的更改等。
- ttread：从终端键盘读入数据。
- ttwrite：向终端显示器写数据。

在此重点介绍ttread和ttwrite。

当从键盘键入数据时，终端中断处理程序把键盘数据放入原始输入缓冲队列，同时送往输出缓冲队列，以便送到显示器输出，当键入回车键时，终端中断处理程序，唤醒所有处于睡眠状态的等待读终端的进程。当一个读终端的进程执行时，由ttread调用行规则程序，从原始输入缓冲队列中依次取出字符，进行必要的格式转换后，放入标准输入缓冲队列，同时，将输出缓冲队列中的字符送往显示器显示，然后把标准输入缓冲队列中的字符复制到应用程序数据存储区。如果当前各原始输入缓冲队列中都没有数据，则ttread睡眠。

对于ttwrite，它首先调用行规则程序，从应用程序数据存储区把要输出的数据送入输出缓冲队列，并对每个输出字符进行行规则处理。当输出缓冲队列中的字符存量大于某个上限时，启动终端输出数据并使写进程睡眠。输出使得输出缓冲队列中的字符存量减少，当输出缓冲队列中的字符存量小于某个下限时，唤醒有关睡眠进程执行写操作。

行规则程序把数据按行（以回车符为结束标志）缓存起来，并按标准格式或原始格式传送数据。行规则程序的功能如下：

- 通过分析将输入字符串组成行。
- 处理删除键，以便用户能够从逻辑上删除已键入的字符。
- 处理行删除键，从而使当前行无效。
- 处理字符格式，例如把制表符变为8个空格序列。
- 把输入的字符回送到显示器。
- 为中断挂起、断线或响应delete键向进程发送软中断信号。
- 允许不做格式变换。

本章小结

本章讲述了 I/O 系统构成、数据传输控制方式、中断技术、缓冲技术、设备分配技术、SPOOLING 系统、I/O 控制过程、Linux 系统的设备管理。

I/O 设备与主机（CPU、内存）之间的通信不是直接的，而通过设备控制器，设备控制器是 I/O 设备和主机之间的接口。在大型计算机系统中，为了使 I/O 操作和计算充分并行，设置了专门负责 I/O 操作控制的通道。

I/O 设备和进程之间的数据传送的控制方式通常有 4 种：程序直接控制方式、中断控制方式、DMA 方式和通道控制方式。程序直接控制方式比较简单而且不需要多少硬件支持，但 CPU 和外设只能串行工作，且 CPU 要花费大量的时间进行循环测试。中断控制方式虽然在一定程度上解决了上述问题，但由于一次数据传输中断次数很多，使得 CPU 要花费很多的时间来处理中断，且能够并行操作的设备总数也受到中断处理时间的限制。DMA 方式和通道控制方式较好地解决了上述问题。这两种方式均采用了外设和内存直接交换数据的方式，只有在一段数据传输结束时，这两种方式才发出中断信号请求 CPU 做善后处理，从而大大减少了 CPU 的工作负担。这两种方式的区别是，DMA 方式要求 CPU 执行设备驱动程序启动设备，给出存放数据的内存地址以及操作方式和传送字节长度等信息，而通道控制方式则是在 CPU 发出 I/O 启动命令后，由通道指令来完成这些工作。

中断技术是计算机系统中普遍采用的一种关键技术，引入这一技术的主要原因是使 CPU 和设备并行工作。

缓冲技术有多种用途。缓冲的设置方式有硬缓冲和软缓冲，缓冲技术分为针对某进程/作业的专用缓冲机制和针对整个系统的公用缓冲机制。专用缓冲机制有单缓冲、双缓冲、循环缓冲，公用缓冲机制有公用缓冲池。

设备分配与很多因素相关。本章就设备的 3 种类型：独享设备、共享设备及虚拟设备介绍了独享分配、共享分配和虚拟分配技术。SPOOLING 技术通常用于多道批处理系统，采用了通道技术、中断技术、多道程序设计技术、虚拟设备技术，使外围设备和主机充分并行操作，大大提高了主机和设备的利用率。

I/O 控制过程从用户进程提出 I/O 请求开始，包括 3 个步骤：响应用户进程的 I/O 请求、设备驱动、设备中断处理。其中关键步骤是设备驱动，设备驱动程序是驱动物理设备和 DMA 控制器或 I/O 控制器进行 I/O 操作的子程序集合，直接与硬件打交道。设备驱动程序与设备中断处理程序配合工作，完成用户进程交付的整个 I/O 操作。

Linux 系统设备管理的特点是把设备分为块设备和字符设备，将设备作为特殊文件，用户可以像使用普通文件一样使用设备文件。

习 题 六

6-1 设备通常有哪几种分类方法？可以分为哪几种类型？

6-2　数据传输控制方式有哪几种？各有什么特点？

6-3　引入中断技术的原因有哪些？

6-4　中断源通常有哪些？

6-5　引入缓冲技术的原因有哪些？

6-6　单缓冲与双缓冲的区别是什么？

6-7　分别解释什么是独享设备和共享设备。

6-8　什么是设备独立性？为什么要引入设备独立性？

6-9　什么是虚拟设备技术？

6-10　SPOOLING 系统采用了哪些技术？

6-11　SPOOLING 系统由哪些部分构成？

6-12　I/O 控制过程包括哪几个步骤？

6-13　访问磁盘的时间由哪几部分构成？

6-14　磁盘调度算法有哪些？它们之间有何区别？

6-15　Linux 系统中磁盘高速缓存的原理是什么？

6-16　Linux 把外部设备分为哪几类？它们的物理特性有何不同？它们的作用有何不同？

6-17　Linux 设备管理的主要特点是什么？

第 7 章　Linux 网络基础

本章学习目标

Linux 是一个网络操作系统，系统管理的很大一部分工作与网络有关。本章将通过一些基本网络操作命令的介绍和对网络配置文件的分析，帮助读者更好地理解网络的工作原理，提高管理技能。主要有以下内容：

- TCP/IP 协议
- 网络配置
- 基本网络服务
- 网络安全

7.1　TCP/IP 协议简介

7.1.1　网络概述

在信息化社会中，计算机已从单一使用发展到群集使用。越来越多的应用领域需要计算机在一定的地理范围内联合起来进行群集工作，从而促进了计算机和通信技术的紧密结合，形成了计算机网络。

在计算机网络中，为了实现计算机之间的相互通信，有许多不同的通信规程和约定，这些规程和约定就是计算机网络协议。这就要求不同的计算机厂商在开发和研制自己的网络系统，设计自己的网络协议时，必须遵循一个统一的约定。国际标准化组织在 1981 年颁布了开放系统互联参考模型，即 ISO/OSI 参考模型，如图 7-1 所示，现在绝大多数网络协议都是参考这一标准设计的。

TCP/IP 于 20 世纪 70 年代末开始被研究和开发，现在已广泛应用于各种网络中，不论是局域网还是广域网都可以用 TCP/IP 协议来构造网络环境。同时 TCP/IP 协议也是 UNIX、Linux 等操作系统中最重要的网络协议。以 TCP/IP 为核心协议的 Internet 更加促进了 TCP/IP 的应用和发展，已成为事实上的国际标准。

从体系结构来看，TCP/IP 协议是 OSI 参考模型七层结构的简化，它只有应用层、传输层、网际层和网络接口层，参见图 7-1。其中，网络接口层对应于 OSI 参考模型的物理层和数据链路层；网际层与 OSI 参考模型的网络层相对应；传输层包含 TCP 和 UDP 两个协议，与 OSI 参考模型的传输层相对应；应用层包含了 OSI 参考模型的会话层、表示层和应用层的功能，规定了与特定的网络应用相关的应用协议，如远程登录、文件传送及电子函件等。

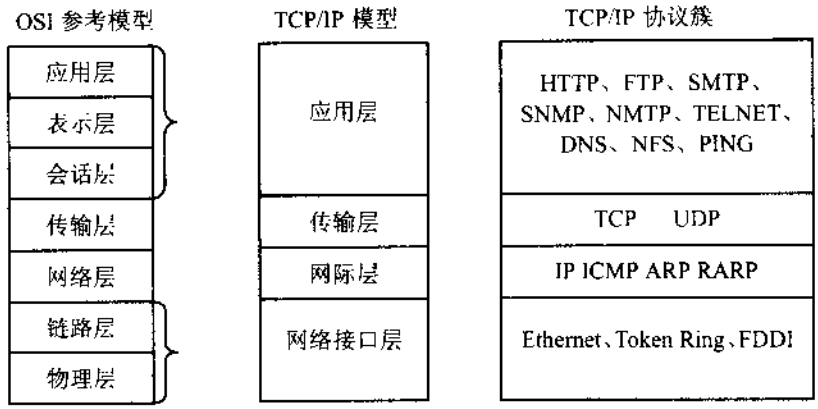


图 7-1　OSI 参考模型和 TCP/IP 协议对应关系

7.1.2　TCP/IP 协议

TCP/IP 协议是由一组通信协议组成的协议簇，其核心部分包括传输层协议（TCP 和 UDP）、网络层协议（IP）和网络接口层，这三层在操作系统内核中实现，而应用层，如 FTP、WWW 等应用程序则通过网络编程接口（socket、Windows Sockets）与核心协议打交道。整个协议簇是一种分层结构，下层为上层提供服务，不同层次之间通过一些接口通信。

1. 以太网

以太网（Ethernet）最初是由美国 Xerox 公司和 Stanford 大学联合开发，并于 1975 年推出的。后来，由 Xerox、Intel 和 DEC 公司合作于 1980 年 9 月第一次公布了 Ethernet 的物理层和数据链路层的技术规范，成为世界上第一个局域网工业标准。IEEE 802.3 国际标准是在 Ethernet 标准的基础上制定的。

以太网工作起来就像一个总线系统，每一台机器都通过一个分接器挂到一根很长的电缆上。为了让机器识别自身，每块以太网卡都有一个由制造商惟一分配的地址（MAC 地址）。当一块以太网卡想要同另一块以太网卡对话时，它向整个以太网电缆发送信息，其中包括自己的 MAC 地址和接收者的 MAC 地址。当两块以太网卡试图在同一时间发送数据时，便会产生冲突。解决这种冲突的办法是两台计算机取消这一次发送，各自等待一段随机的时间，再进行发送数据的尝试。

允许 Linux 和 TCP/IP 在以太网地址和 IP 地址间进行连接的是地址解析协议（ARP）缓存。当一台 Linux 机器开始在以太网上发言时，它会通过 ARP 询问子网上其他机器的 MAC 地址。

2. IP 协议

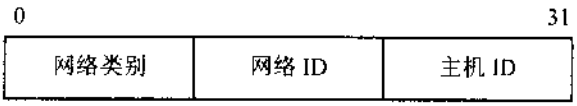
IP 协议位于 TCP/IP 协议簇的第三层，是 TCP/IP 协议的传输系统，也是整个 TCP/IP 协议簇的核心。IP 协议负责数据报在计算机之间寻址，决定数据传送到哪里及出现网络故障时如何更换路由。IP 不保证传送的可靠性，在主机资源不足的情况下，它可能丢弃某些数据报，同时 IP 也不检查被数据链路层丢弃的报文。

（1）IP 地址。TCP/IP 网络要进行通信，每台主机（在 TCP/IP 中，端节点一般称为主机 Host）都要预先分配一个惟一的 32 位地址作为该主机的标识符，这个主机必须使用该地址进行所有通信活动，这个地址称为 IP 地址。IP 地址通常由网络标识 ID（Net）和主机标识 ID（Host）两部分组成，可标识互连网络中任何一个网络中的任何主机。

网络标识也称为网络地址，用于辨别子网，同一子网上的所有 TCP/IP 主机的网络 ID 都相同。

主机标识也称为主机地址，用于辨别每个网络内的主机。

IP 地址格式如下：



其中网络 ID 是大于 1 位的二进制数，主机 ID 也是大于 1 位的二进制数。如计算中心网关地址 132.111.002.001 分解成协议所能认识的地址就是：

网络 ID=132.111

主机 ID=002.001

后面将介绍为什么这样分解。需要注意的是，在 IP 地址中，全 0 主机 ID 代表的是网络，全 1 主机 ID 代表的是广播。比如对 IP 地址 132.111.002.001 而言，它所在的网络用 132.111.002.0 来表示，而 132.111.002.255 则代表向整个 132.111.002 网络广播的地址。

以上的设计贯彻了灵活的原则，给 IP 地址带来了很好的实用性。可以根据机器的数目给不同的单位分配不同的网段，于是就诞生了 IP 地址分类的问题。举个例子来说，像 132.111 网段可以拥有的主机数可以用 16 位主机地址表示，即可以达到 65536 台主机，所以像一些大学，拥有一个 132.111 这样的网段是合适的。对于大型的跨国公司，如 IBM、微软，则可能拥有上十万台的主机，16 位主机地址是不能满足要求的，可以分配一个 24 位的主机地址。对一些小公司而言，可能分配一个拥有 255 台主机数的 8 位主机地址的网段就能满足要求。

IP 地址分类如下：

A 类地址：网络类别字段占 1 位，即第 0 位为 0，表示是 A 类地址，第 1～7 位表示网络地址，第 8～31 位表示主机地址。它所能表示的范围为 0.0.0.0～127.255.255.255，即能表示 126 个网络地址，16 387 064 个主机地址。A 类地址通常用于超大型网络的场合。

B 类地址：网络类别字段占 2 位，即第 0、1 位为“1 0”，表示 B 类地址，第 2～15 位表示网络地址，第 16～31 位表示主机地址。它所能表示的范围为 128.0.0.0～191.255.255.255，即能表示 16 256 个网络地址，64 576 个主机地址。B 类地址通常用于大型网络的场合。

C 类地址：网络类别字段占 3 位，即第 0、1、2 位为“110”，表示是 C 类地址，第 3～23 位表示网络地址，第 24～31 位表示主机地址。它所表示的范围为 192.0.0.0～223.255.255.255，即能表示 2 064 512 个网络地址，254 个主机地址。C 类地址通常用于校

园网或企业网。

此外，还有 D 类和 E 类 IP 地址。D 类地址用于多址广播地址，供特殊协议向选定的节点发送信息时用。E 类地址暂时保留。

IP 地址经过分类，确实方便了记忆和管理，但是也造成了地址的浪费，许多地址即使空着，但归属某个单位或部门，别人没有办法使用，从而造成了地址空间的紧张，当然很大程度上也是因为 32 位的地址空间限制。所以 IPv6（目前的 IP 协议版本为 IPv4）技术的一部分工作就是解决地址空间不足的问题。

在 Internet 中，一台计算机可以有一个或多个 IP 地址，就像一个人可以有多个通信地址一样，但两台或多台计算机不能共用一个 IP 地址。如果两台计算机的 IP 地址相同，则会引起异常现象，无论哪台计算机都将无法正常工作。

（2）子网掩码。通常将一个较大网络分成多个较小的网络，每个小网络使用不同的网络 ID，这样的小网络称之为子网。

在网络通信时，若想找到子网，需定义子网掩码。子网掩码与 IP 地址一样是一个 32 位的值，将它与主机的 IP 地址做按位“与”运算，可以屏蔽一部分 IP 地址，从而确定出这个网络地址。子网掩码可概括为如下两个功能：用于区分出网络地址和主机地址；用于将网络分割为多个子网。

例如，某主机 A 的 IP 地址为 192.166.008.002，子网掩码为 255.255.255.0，将这两个数据做“与”运算后，得出值中非 0 部分即为网络地址，即 192.166.008，剩余字节为主机地址，也就是 002。若有主机 B 的 IP 地址为 192.166.008.005，子网掩码同主机 A，通过子网掩码发现主机 B 与主机 A 网络地址相同，因此两台主机在同一个网络上。当主机 A 向主机 B 发送数据包时，就可以将数据包直接发送给主机 B。

根据子网掩码的工作原理，还可以设计可变长度子网掩码 VLSM，从而将大的网段划分成小的网段，也可以用小的网段构建大的网段（超网），不同单位可以根据需要自行决定采用什么方式工作。

3. TCP 协议

TCP 协议是是一种面向连接的、可靠的传输层协议。面向连接是指一次正常的 TCP 传输需要通过在 TCP 客户端和 TCP 服务端建立特定的虚电路连接来完成，该过程通常被称为“三次握手”。可靠的传输协议可避免数据传输错误。ISO/OSI 传输层标准中的第四类 TP-4 就是以 TCP 为原型而制定的。TCP 协议能够向发送方提供到达接收方的数据包的传送信息。在传送信息的过程中出现数据丢失时，TCP 协议可以重新发送丢失的数据包，直到发送成功或出现网络超时。在分组发生丢失、破坏、重复、延迟或失序的情况下，TCP 服务通过一种可靠的进程间通信机制能够自动纠正各种差错，使网络环境得到优化。如果传送数据的速度大大快于接收方接收数据的速度，TCP 协议可以采用数据流控制机制减慢数据的传送速度，协调发送方和接收方的数据响应。TCP 协议可以支持许多高层协议 ULP（Upper Level Protocol），它对高层协议的数据结构无任何要求，只将它们作为一种连续的数据流。

4. UDP 协议

UDP 也是常用的传输层协议，提供非面向连接的、不可靠的数据流传输服务。这种服

务不确认报文是否到达，不对报文排序，也不进行流量控制，因此 UDP 报文可能会出现丢失、重复和失序等现象。与 TCP 相同的是，UDP 协议也是通过端口号支持多路复用功能，但是不能建立连接，而是向目标计算机发送独立的数据包。

UDP 是一种简单的协议机制，通信开销很小，效率比较高，适合于对可靠性要求不高，但需要快捷、低延迟通信的应用场合，如多媒体通信等。

7.2 配置 Linux 网络

本节主要介绍在 Red Hat Linux 环境下如何配置网络。

7.2.1 网络配置文件

在 Linux 系统中，TCP/IP 网络是通过若干个文本文件进行配置的，了解这些配置文件对于理解一些配置命令和界面配置工具大有好处。可以通过编辑这些文件来完成联网工作。下面将介绍系统中重要的 TCP/IP 网络配置文件。

1. /etc/HOSTNAME 文件

该文件包含系统的主机名称和完全的域名，如：

linux.cic.tsinghua.edu.cn（Linux 为主机名，cic.tinsghua.edu.cn 为域名）

2. /etc/host.conf 文件

该文件指定如何解析主机名。Linux 通过解析器库获得主机名对应的 IP 地址。示例如下：

order bind,hosts

multi on

nospoof on

order bind,hosts 指定主机名查询顺序，这里规定先使用 DNS 解析域名，然后再查询 /etc/hosts 文件。

multi on 表示/etc/hosts 文件中指定的主机可以有多个地址，拥有多个 IP 地址的主机一般称为具有多个网络接口。

nospoof on 指不允许对该服务器进行 IP 地址欺骗。IP 欺骗是一种攻击系统安全的手段，通过把 IP 地址伪装成别的计算机，来取得其他计算机的信任。

3. /etc/services 文件

/etc/services 包含了服务名和端口号之间的映射，不少系统程序要使用这个文件，下面是 Red Hat 安装时默认的/etc/services 中的前几行：

```
tcpmux 1/tcp # TCP port service multiplexer
echo 7/tcp
echo 7/udp
discard 9/tcp sink null
discard 9/udp sink null
systatll/tcp users
```

最左边一列是主机服务名，中间一列是端口号，“/”后面是端口类型，可以是 TCP

也可以是 UDP。任何后面的列都是前面服务的别名。在这个文件中也存在别名，它们出现在端口号后面，在上述例子中，sink 和 null 都是 discard 服务的别名。

4. /etc/sysconfig/network 文件

该文件用来指定服务器上的网络配置信息，包含控制与网络有关的文件和守护程序行为的参数。下面是一个示例：

```
NETWORKING=yes
HOSTNAME=machine1
DOMAINNAME=cic.tsinghua.edu.cn
GATEWAY=210.34.6.2
FORWARD_IPV4=yes
GATEWAYDEV=eth0
```

其中：

```
NETWORK=yes/no          网络是否被配置
HOSTNAME=hostname       hostname 表示服务器的主机名
DOMAINNAME=domainname   domainname 分配给该机器的域名，这个字段基本不用
GATEWAY=gw-ip           gw-ip 表示网络网关的 IP 地址
FORWARD-IPV4=yes/no     是否开启 IP 转发功能
GATEWAYDEV=gw-dev       gw-dev 表示网关的设备名
```

注意：因为不同版本的系统或不同的应用软件会从不同的地方去取主机名，为了保证同一个系统的所有应用程序采用的是相同的主机名，应保持 HOSTNAME 文件与该文件中的主机名一致。

5. /etc/sysconfig/network-scripts/ifcfg-ethN 文件

在 Red Hat 中，系统网络设备的配置文件保存在/etc/sysconfig/network-scripts 目录下，ifcfg-eth0 包含第一块网卡的配置信息，ifcfg-eht1 包含第二块网卡的配置信息。

下面是/etc/sysconfig/network-scripts/ifcfg-eth0 文件的示例：

```
DEVICE="eth0"                  /*表示物理设备的名字*/
IPADDR="192.168.2.1"           /*表示赋给该卡的 IP 地址*/
NETMASK=" 255.255.255.0"       /*表示网络掩码*/
NETWORK="192.168.2.0"          /*表示网络地址*/
BROADCAST=" 192.168.2.255"     /*表示广播地址*/
ONBOOT="yes"                   /*启动时是否激活该卡，值为 yes/no*/
BOOTPROTO="none"
```

proto 取值可以是：

（1）none：无须启动协议。

（2）bootp：使用 bootp 协议。

（3）dhcp：使用 dhcp 协议。

USERCTL="no"　/*表示是否允许非 root 用户控制该设备，值为 yes/no*/

若希望手工修改网络地址或在新的接口上增加新的网络界面，可以通过创建新的文件，文件名为 ifcfg-ethN，N 为 0、1 等数字。

6. /etc/hosts 文件

/etc/hosts 文件包含了 IP 地址和主机名之间的映射，还包括主机名的别名，IP 地址的

设计使计算机容易识别，但却很难记住它们，为了解决这个问题，创建了/etc/hosts 文件。下面是一个例子文件：

```
127.0.0.1 machine1 localhost.localdomain localhost
192.168.1.100 machine7
192.168.1.101 otherpc otheralias
```

在这个例子中，本机名是 machine1，otherpc 还有别名 otheralias，它可以指向 otheralias。一旦配置完机器的网络配置文件，应该重新启动网络以使修改生效，使用下面的命令来重新启动网络：

/etc/rc.d/init.d/network restart

/etc/hosts 文件通常含有主机名、localhost 和系统管理员经常使用的系统别名，有时 telnet 到 Linux 机器要等待很长时间，可以通过在/etc/hosts 加入客户机器的 IP 地址和主机名的匹配项减少登录等待时间。在没有域名服务器的情况下，系统上所有网络程序都通过查询该文件来解析对应于某个主机名的 IP 地址，否则，其他的主机名通常使用 DNS 来解决，DNS 客户部分的配置在文件/etc/resolv.conf 中。

7. /etc/resolv.conf 文件

该文件是由 DNS 客户端解析器（resolver，一个根据主机名解析 IP 地址的库）使用的配置文件，它包含主机的域名搜索顺序和 DNS 服务器的地址。示例如下：

```
search cic.tsinghua.edu.cn
nameserver 166.111.4.5
nameserver 166.111.8.28
```

“search 域名”表示当提供了一个不包括完全域名的主机名时，在该主机名后添加域名后缀；nameserver 表示解析域名时使用该地址指定的主机为域名服务器，其中域名服务器是按照文件中出现的顺序来查询的。

8. /etc/init.d/network 主机地址、子网掩码和网关

不像很多其他 UNIX 和 Linux 操作系统，Red Hat 当前并不能自动通过/etc/hostname 和/etc/hosts 文件来配置网络。为了改变主机默认的 IP 地址，必须直接编辑/etc/init.d/network 脚本使其反映正确的网络配置。这个文件包括声明 IP 地址、掩码、网络、广播地址和默认路由器的变量。下面是这个文件中相应的部分：

```
IPADDR=192.168.1.100
NETMASK=255.255.255.0
NETWORK=192.168.1.0          /*表示网络地址*/
BROADCAST=192.168.1.255
GATEWAY=192.168.1.1
```

7.2.2 手工配置 TCP/IP 网络

1. 设置主机名

一般情况下，网络应用程序依赖于本地主机名，这是在系统启动时由 hostname 命令完成的，即#hostname 主机名。

2. 设置 IP 地址

当用户在自己的主机上进行单机操作时可以忽略这一步，因为只需要回送接口的 IP 地址，而这个接口的 IP 地址始终都是 127.0.0.1。

如果要连接到一个网络中，必须要求该网络的管理员分配一个 IP 地址。如果接入的网络不是因特网，用户可以根据自己的喜好选择一个合法的网络地址。但是，如果接入因特网，就必须想办法获得一个正式的 IP 地址。

如果想同时操作若干个以太网，就必须将自己的网络分为若干个子网。

3. 主机和网络文件

将自己的网络分成若干个子网后，应该利用 etc/hosts 文件进行简单的主机名解析。通常情况下，可把所有的本地主机名和地址放入主机文件内。

查询主机 IP 地址时，如果要所有应用程序专用/etc/hosts 文件，必须对/etc/host.conf 文件进行编辑。在以关键字 order 开头的所有代码行前加上一个“#”号，就可以批注所有行，并插入下面这行：

order hosts

主机文件中每行都有一个条目，由 IP 地址、主机名和主机别名清单组成，这些字段用空格或标号隔开，而且地址字段必须在第一列内，跟在“#”后面的被视为批注。

和 IP 地址一样，应该用一个好记的名字代表某个网络，所以还要有一个/etc/networks 文件，该文件指明网络名和网络编号之间的对应关系。

4. 配置网络接口

配置网络接口采用的基本工具是 ifconfig 和 route。

（1）ifconfig 命令。ifconfig 用于命令接口能够被内核联网层访问。要想启用接口，使用下面的命令：

ifconfig interface ip-address

此代码是将 ip-address 分给接口，就激活了它，其他参数都采用默认值。

注意：用 ifconfig 命令设置的参数会在机器重新启动时丢失。可以在启动网络的脚本文件中添加以上这些命令，这样每次重启机器时系统都会执行这些脚本文件，达到永久改变的目的。如果要永久改变 IP 地址，可改变/etc/sysconfig/network-scripts/ifcfg-ethN 文件中 IPADDR 项的内容，如：

IPADDR="192.166.002.254"

（2）route 命令。route 命令用于在内核路由表内增加还是删除路由，由管理员手工维护。对于一般主机而言，它可以告诉一台主机如何发送数据给其他主机。如果主机充当路由器，它还要负责在不同网络间转发数据。路由是在 IP 层进行的，当收到一个目的地为其他主机的报文后，本主机将其网络地址与路由表中的内容进行比较，如果目的地为直接相连的网络，则直接将报文发送至该网络；如果没有直接匹配的网络地址，则将报文发送给某个默认路由器，由它进一步路由报文；如果没有默认路由，将返回给发送端一个网络不可到达的错误信息。除了由管理员手工修改路由表信息外，主机本身也可以收集 RIP（Routing Information Protocol）路由广播或其他 IGP（Interior Gateway Routing Protocol）

类路由协议的广播信息，生成动态路由信息。

使用下面的命令调用它：

#route [add|del] target

其中，add 和 del 用于判断是增加还是删除路由。

5. 配置回送接口

首先激活的接口是回送窗口，如下所示：

#ifconfig th 127.0.0.1

ifconfig 将查找主机文件内的主机名，其中一个条目将该主机名声明为 127.0.0.1 的主机名：localhost 127.0.0.1。

如果要查看接口的配置情况，将接口名作为 ifconfig 的参数，使用#ifconfig th 调用它即可。

由此可知，回送接口已经分到一个网络掩码 255.0.0.0，这是因为 127.0.0.1 是 A 类地址。

现在可以运行网络了，但是还缺一项，即路由表中的一条，它告诉 IP 可以使用这个接口作为去往目标 127.0.0.1 的路由，可通过使用下面的命令完成：

#route add 127.0.0.1

这样，就可以在一台独立主机上使用连网应用程序了。

6. 配置以太网接口

以太网接口的配置和回送接口的配置大致相同，不过使用子网技术时，以太网接口的配置要求的参数比后者要多一些。

使用下面的命令调用 ifconfig：

ifconfig eth0 192.166.002.004 netmask 255.255.255.0

这条命令自动指定了广播地址，网络地址可以通过 IP 地址与子网掩码按位“与”获得，主机地址全为 1，所以得到广播地址为 192.168.2.255。

接下来的情形和回送接口类似。

7.3 基本网络服务介绍

7.3.1 域名服务器（DNS）

连接 TCP/IP 的每个网络接口用一个惟一的 32 位的 IP 地址标识，但由于数字比较复杂，难以记忆，而且没有形象性，因而发明了域名系统，在这种情况下，可以使用易于理解和较为形象的名称作为一台计算机的标识。大多数情况下，数字地址和域名地址可以交替使用，但无论用数字地址或是域名进行网络应用时，总是以 IP 地址为基础进行的。在网络进行连接前，系统必须将域名地址转换成 IP 地址，这就是 DNS 的任务。

大多数 DNS 服务器，包括 Red Hat Linux 附带的版本，都使用 Bind （Berkeley Internet Name Domain），当前是 Bind 8 版本。DNS 提供了从名字到 IP 地址的映射关系，这种映射关系不必是一一映射，一个 IP 可以有多个域名，一个域名也可以对应多个 IP。

如果要为站点配置 DNS，需要如下工具。

1. named 程序

named 是在 DNS 上运行的守护程序，用来处理查询和负责执行区间传输。如果它不能对某一查询做出回答，它将负责把这一请求转发给可以解答的服务器。通过区间传输，可以在 Internet 上传播修改过的 DNS 信息。

2. 解析程序库

为了使用和测试 named，要通过在/etc 目录下创建 resolve.conf 文件，把 Linux 设置在使用 DNS 状态。这个文件的用途是设置域名服务器的顺序，如果在第一个域名服务器上找不着某个域名，它就会到第二个域名服务器上去寻找，依次类推。服务器也可通过网关接入 Internet，而它的 DNS 只包含本地网段的主机信息，因此，在连入 Internet 后，在本地找不着某个域名时就必须到别的 DNS 服务器上去查找。

3. traceroute 程序

这个程序是用来确定数据包从当前网络传输到其他网络所采用的路径，对调试网络连接非常有用，特别是怀疑在其他网络中发生故障时。

4. nslookup 程序

nslookup 是用来确保解析程序和 DNS 服务器正确配置，这一任务是通过主机名解析 IP 地址或把一个 IP 地址解析为域名来完成的。

7.3.2　Apache 服务器

Web 服务器是互联网上人们使用最多的一种服务器，它已经成了当今社会不可缺少的一种信息传播方式。目前市场上最流行的运行于 Linux 上的 Web 服务器是 Apache 服务器。

纵观 Apache，它为网络管理员提供了丰富多彩的功能，包括目录索引、目录别名、内容协商、可配置的 HTTP 错误报告、CGI 程序的 SetUID 执行、子进程资源管理、服务器端图像映射、重写 URL、URL 拼写检查以及联机手册 man 等。也就是说，如果在 Linux Server 上成功安装配置了 Apache 后，您的计算机也将随着 Apache 的生效而摇身一变，成为一台名副其实的 Web Server，这种变化的确是激动人心的。

与 Apache 配置相关的文件共有如下 3 个。

1. httpd.conf 配置文件

作用：定义 HTTP 协议的设置值和针对该服务器的操作。下面介绍一些配置命令。

（1）SeverType。定义 WebServer 的启动方式为 standalone，以增强其对大量访问的及时响应性。

（2）ServerRoot。指定服务器目录的绝对路径，即 Apache 服务器的 home 目录。

（3）ServerName。定义客户端从服务器读取数据时返回给客户端的主机名，其默认值是 localhost。

（4）DocumentRoot。设置所有 Apache 所提供的文档的根目录。

（5）DirectoryIndex。设置多种成功访问主页的方式，以提高系统的容错性。

（6）Port 端口号。定义服务器所使用的 TCP 的端口号。

（7）ServerAdmin root@localhost。设置 Web 管理员的邮件地址。

2. Access.conf 文件

负责基本的读取文件控制，限制目录所能执行的功能及访问目录的权限设置。

格式：

```
<Directory /home/httpd/html>
-order deny,allow
-deny from all
-allow from localhost
-AllowOverride None
-Options ExecCGI
-</Directory>
```

deny，拒绝访问；allow，允许谁可以访问；none，表示除了观看以外不能做其他任何事情；options，定义所能执行的操作；ExecCGI，表示允许执行 CGI。

3. Srm.conf 文件

数据配置文件，在这个文件中主要设置 WWW Server 读取文件的目录、目录索引时的画面、CGI 执行时的目录等。

格式：

```
-DocumentRoot /home/httpd/html
-UserDir public_html
-DirectoryIndex index.html index.shtml index.cgi
```

–FancyIndexing on：如果没有找到 index.html 等文件，则创建一个 html 目录清单，列出该目录所有文件，包括最后修改时间等内容，否则只有一个文件列表。

–AddIcon：设置目录索引上文件旁边的图标。

–ReadmeName README：文件内容加到索引底部。

–HeaderName HEADER：文件内容加到索引头部。

–IndexIgnore .??* *~ *# HEADER* README* RCS：制作索引要忽略的文件类型，以空格分开。

–AccessFileName .htaccess：设置控制目录访问权限的文件的文件名。

–TypesConfig /etc/mime.types：设置 mime.types 文件在系统中的位置。

–DefaultType text/plain：默认（或未知）文件类型。

–AddEncoding x-compress Z

–AddEncoding x-gzip gz：允许浏览器浏览压缩文件。

–AddLanguage en .en：设置文档语言（多语言适应）。

–LanguagePriority en fr de：优先级。

–Redirect fakename url：重定向。例如 Redirect beijing http://somewhere.com/maps/beijing 对 http://somewhere.com/beijing/index.html 的请求被重定向到 http://somewhere.com/maps/beijing。

–Alias /icons/ /home/httpd/icons/：定义别名。

–ScriptAlias /cgi-bin/ /home/httpd/cgi-bin/：定义脚本目录的别名。

–AddType type/subtype ext1：添加 Mime 类型，例如 AddType application/x-httpd-php3.php3。

–AddHandler action-name ext1：定义 CGI 脚本的格式，约定以.cgi 为后缀，例如 AddHandler cgi-script.cgi。

–Action media/type/cgi-script/location：定义媒体类型，任何时候一个匹配的文件被调用时将执行相应的脚本。

–MimeMagicFile /etc/httpd/conf/magic：允许服务器通过文件本身的信息来确定文件类型。

7.3.3　邮件服务器

自从加州大学伯克利分校 U.C Berkeley 完成 sendmail 的最初版本以来，sendmail 受到了业界长久的重视，从 1983 年的 V5 版本一直到 1993 年重新改写的 V8 版本，都受到了人们的尊重。

sendmail 为 Linux 提供 SMTP 连接所需的服务，它对邮件信息进行分析并把它传送到目的地。sendmail 采用开放源码的开发方式编写，其所需源码都可以免费得到并自由发布。它只提供了邮件路由功能，将发送留给管理员可以选择的本地代理。

与 sendmail 配置相关的配置文件有 sendmail.cf、sendmail.cw、sendmail.hf、sendmail.st。下面主要介绍一下 sendmail.cf 文件。

/etc/sendmail.cf 是 sendmail 的主配置文件。

功能：控制 sendmail 运行时的配置，可以指定用来改写寄件人和收件人的地址及选择邮件发送器的规则集。

完整的 sendmail 配置应该包括以下 7 部分。

Local Info（本地信息）：这部分定义了本地主机的信息。

Options（选项）：用来设置定义 sendmail 环境的选项。

Message Precedence（消息的优先级）：sendmail 消息的优先级。

Trusted Users（信任用户）：定义发送邮件时允许改变发送地址的用户。

Format of Headers（头格式）：定义在 sendmail 中插入的邮件头信息。

Rewriting Rules（改写规则）：这部分保存着改写邮件地址命令，使用该命令可以将邮件地址从用户邮件程序的地址形式改写为邮件发送程序所需要的地址形式。

Mailer Definition（邮寄者说明）：定义发送邮件的程序，改写规则是邮件者使用的规则，在本部分定义。

一般格式：配置文件的每一行都以一个单一的命令字开头，这个字符用来指明这行的功能和语法。以“#”开始的行是注释行；空行被忽略；以空格或制表符开始的行是上一行的续行，不过通常应避免续行。

命令字：

（1）D 操作符——宏。

宏类似于 shell 变量。当为一个宏定义了值后，可以在配置文件的后面对它进行引用，宏将会被它的值所取代。宏可以多于一个字符。Sendmail 使用小写字母定义自己的宏，大写字母可以自由使用。

（2）C 和 F 操作符——类。

类和宏类似，但是在改写规则中用于不同的目的。与宏一样，类的名字也是单一的字符，小写字母由 sendmail 使用，大写字母用于由用户定义的类。

（3）H 操作符——邮件头的定义。

一般不需要对配置文件提供的邮件头进行修改。下面是一个邮件头的示例：

```
H?P?Return-Path: <$g>
HReceived: $?sfrom $s $.$?_($?s$|from $.$_)
H?D?Resent-Date: $a
H?D?Date: $a
H?F?Resent-From: $?x$x <$g>$|$g$.
H?F?From: $?x$x <$g>$|$g$.
H?x?Full-Name: $x
H?M?Resent-Message-Id: <$t.$i@$j>
H?M?Message-Id: <$t.$i@$j>
```

（4）K 操作符——关键字文件。

sendmail 使用了一些关键字数据库。K 操作符可用来向 sendmail 通知数据库的位置、类别以及访问方法。

（5）M 操作符——邮件发送器的定义。

邮件发送器或者是 MTA 或者是最终投递代理。针对每种目的，可以定义一个专门的邮件发送器。

（6）O 操作符——设置选项。

sedmail 具有许多选项，这些选项主要用来改变 sendmail 的操作或用来通知它所使用的文件的位置。大部分选项既可以通过命令行提供，也可以在配置文件中给出。要在命令行中指定别名文件，可以使用 O 选项。

（7）P 操作符——邮件的优先级。

可以在邮件中包含标题，用来指定邮件中相对比较重要的内容。sendmail 可以利用这些标题来确定信件的优先级。

（8）R 和 S 操作符——规则集和改写规则。

规则集用来探测错误的地址，把地址改写为远程邮件发送器能够理解的形式，并把邮件路由到 sendmail 内部的某个邮件发送器。

规则集由 S 命令声明，跟在它后面的是用来标识规则集的编号。sendmail 收集后续的 R 行，直到发现另一个 S 操作符或配置文件的结尾。

（9）V 操作符——sendmail.cf 的版本级别。

通知 sendmail 在配置文件中可以找到那些特性。

其实，配置一个完整的 sendmail 服务器是一件极其复杂的工程，然而对于我们所适用的范围来说，很多功能是没有必要的，没有被提出的配置一般情况下不需要管理员考虑，按照默认的做就可以了。

7.3.4 samba 服务器

可以把 samba 当成是一个局域网络上的 file/printer sever，它可以提供文件系统、打印

机或其他信息，并与 samba sever 在同一个子网域的 samba client 共享。

samba 是用来实现 SMB 的一种软件，它的工作原理是让 NETBIOS（Windows 95 网络邻居的通讯协议）和 SMB（Server Message Block）这两个协议运行于 TCP/IP 通信协议之上，并使用 Windows 的 NETBEUI 协议让 Linux 计算机可以在网络邻居上被 Windows 计算机看到。所以，通过 samb，Linux 和 Windows 可以在网络邻居上沟通，互相浏览共享的文件。

安装完 samba 后，可以看见/etc/samba 目录，里面存放和 samba 相关的一些文件，最重要的是 smb.conf，现在根据配置文件介绍它的配置选项。

（1）workgroup = MYGROUP。定义该 Samba 服务器所在的工作组或域（如果下面的 security=domain）。

（2）server string = MY Samba Server。设定机器的描述，当通过网络邻居访问时可以在备注里看见这个内容，而且可以使用 samba 设定变量。

（3）hosts allow = 网络或主机。控制可以访问共享的主机和子网及设置允许访问的网络和主机 IP，例如允许 192.168.1.0/24 和 192.168.2.1/32 访问，就用 host allow = 192.168.1. 192.168.2.1 127.0.0.1（网络注意后面加“.”号，各个项目间用空格隔开，本机也要加进去）。

（4）printcap name = printcapFile。到 printcapFile（一般是/etc/printcap）这个文件中取得打印机的描述信息。

（5）load printers = yes|no。设定是否自动共享打印机，默认值为 yes。

（6）printing = PrintSystemType。定义打印系统的类型，默认是 lprng，可选项有 bsd、sysv、plp、lprng、aix、hpux、qnx。

（7）guest account = pcguest。定义游客帐号，而且需要把这个帐号加入/etc/passwd，不然它就用默认的 nobody。

（8）encrypt passwords = yes|no。设置是否对密码进行加密，samba 本身有一个密码文件/etc/samba/smbpasswd，如果不对密码进行加密，则在验证会话期间客户机和服务器之间传递的是明文密码，samba 直接把这个密码和 Linux 中的/etc/samba/smbpasswd 密码文件进行验证。默认值为 no。

（9）smb passwd file。设置存放 samba 用户密码的文件 smbPasswordFile（一般是/etc/samba/smbpasswd）。默认值为 smbPasswordFile

（10）path。指定共享的路径，可以配合 samba 变量使用。

（11）valid users。指定能够使用该共享资源的用户和组。

（12）invalid users。指定不能够使用该共享资源的用户和组。

（13）public 和 guest ok。指明该共享资源是否能被游客帐号访问，这个开关有时候也叫 guest ok，所以有的配置文件中出现 guest ok = yes 其实和 public = yes 是一样的。

（14）security = security_level。定义 Samba 的安全级别，按从低到高分为 4 级：share、user、server、domain，它们对应的验证方式如下。

share：没有安全性的级别，任何用户都可以不要用户名和口令访问服务器上的资源。

user：samba 的默认配置，要求用户在访问共享资源之前资源必须先提供用户名和密码

进行验证。

server：和 user 安全级别类似，但用户名和密码是递交到另外一个服务器去验证，如递交给一台 NT 服务器。如果递交失败，就退到 user 安全级。

domain：这个安全级别要求网络上存在一台 Windows 的主域控制器，samba 把用户名和密码递交给它去验证。

后面三种安全级别都要求用户即使在本 Linux 机器上也需要系统帐户，否则不能访问的。

（15）password server = <NT-Server-Name>。当前面的 security 设定为 server 或 domain 时才有必要设定它。

（16）password level 和 usename level。这些选项决定大小写无关的用户名和口令比较的级。默认值为 0，表明客户端提供的口令或用户名首先与服务器上的记录进行大小写比较，如果失败，则客户端的用户名或口令转换为小写，然后和服务器上的记录进行比较。

7.4 Linux 网络安全

7.4.1 Linux 系统的文件安全

Linux 文件系统是由文件和目录构成的树形结构，每个文件目录包括文件类型、文件的存取权限、文件名、文件所有者、文件大小、文件建立及修改的日期等内容。任何一项内容遭受未授权的修改，文件的安全性都将遭到破坏。保护文件系统的安全性应该从以下几个方面入手。

1. 文件存取权限的设置

文件属性决定了文件的被访问权限，即谁能存取或执行该文件。用 ls-l 可以列出详细的文件信息，如：

-rwxr-xr-- 1 wang group 600 99 Jul 28 21:12 file 包括了文件许可、文件联结数、文件所有者名、文件相关组名、文件长度、上次存取日期和文件名。

其中，文件许可权限分为四部分：

-：表示文件类型。

第一个 rwx：表示文件属主的访问权限。

第二个 r-x：表示文件同组用户的访问权限。

第三个 r--：表示其他用户的访问权限。

改变许可方式可使用 chmod 命令，并以新许可方式和该文件名为参数。新许可方式以 3 位 8 进制数给出，r 为 4，w 为 2，x 为 1，如 rwxrwxr--为 774。

chmod 也有其他方式的参数可直接对某组参数修改，在此不再多说。文件许可权限可用于防止偶然性地重写或删除一个重要文件（即使属主是自己）。

改变文件的属主和组名可用 chown 和 chgrp，但修改后原属主和组员就无法修改回来了。

2. 设置用户 ID 和同组用户 ID 许可

用户 ID 许可（SUID）设置和同组用户 ID 许可（SGID）设置只给予可执行的目标文件。（只有可执行文件才有意义）当一个进程执行时就被赋予 4 个编号，以标识该进程隶属于谁，分别为实际和有效的 UID，实际和有效的 GID。有效的 UID 和 GID 一般和实际的 UID 和 GID 相同，有效的 UID 和 GID 用于系统确定该进程对于文件的存取许可，而设置可执行文件的 SUID 许可将改变上述情况，当设置了 SUID 时，执行此 SUID 的进程的有效 UID 被设置为程序拥有者的 UID，而不是执行该程序的用户的有效 UID，因此，由该程序创建的都有与该程序所有者相同的存取许可。这样，程序的所有者将可通过程序的控制在有限范围内向用户发表不允许被公众访问的信息。同样，SGID 是设置有效 GID。

用 chmod u+s 文件名和 chmod u-s 文件名来设置和取消 SUID 设置。用 chmod g+s 文件名和 chmod g-s 文件名来设置和取消 SGID 设置。

当文件设置了 SUID 和 SGID 后，chown 和 chgrp 命令将全部取消这些许可。

7.4.2　用户口令安全

每个 Linux 的用户都拥有一个帐号，通过登录到这个帐号才能有限制地使用系统，而保护自身文件安全的惟一屏障就是口令，一旦这道屏障被破坏，整个系统的安全便无法得到保证。

Linux 系统中的/etc/passwd 文件含有全部系统需要知道的关于每个用户的信息（加密后的口令也可能存于/etc/shadow 文件中）。

/etc/passwd 中包含用户的登录名，以及经过加密的口令、用户号、用户组号、用户注释、用户主目录和用户所用的 shell 程序。其中，用户号（UID）和用户组号（GID）用于 Linux 系统惟一地标识用户主和同组用户及其他用户的访问权限。

/etc/passwd 中存放的加密口令用于与用户登录时输入的口令经计算后进行比较，符合则允许登录，否则拒绝用户登录。用户可用 passwd 命令修改自己的口令，不能直接修改/etc/passwd 中的口令部分。

一个好的口令应至少有 6 个字符长，不要取用个人信息，如生日、名字、反向拼写的登录名、房间中可见的东西，普通的英语单词也不好（因为可用字典攻击法），口令中最好有一些非字母，如数字、标点符号、控制字符等，还要好记一些，不能写在纸上或计算机中的文件中，选择口令的一个好方法是将两个不相关的词用一个数字或控制字符相连，并截断为 8 个字符。当然，如果能记住 8 位乱码自然更好。

不应将同一个口令在不同机器中使用，特别是在不同级别的用户上使用同一口令，会引起全盘崩溃。用户应定期改变口令，至少 6 个月改变一次，系统管理员可以强制用户定期做口令修改。

为防止眼明手快的人窃取口令，输入口令时应确认无人在身边。

7.4.3　防火墙技术

1. 防火墙的概念

以前，人们经常在木屋和其他建筑物之间修筑一道砖墙，以便在火灾发生时阻止火势

蔓延到其他建筑物，这种砖墙被称为防火墙。后来，这种防火墙的保护机制被引入到计算机网络安全技术上，以保护计算机网络免受外部入侵者的攻击。防火墙是一种网络安全产品，它提供了一种保护机制，其目的是保护一个网络不受来自另一个不可信任网络的非法侵入，它对网络的保护主要体现在防止非法的外部用户侵入 Intranet 访问资源和窃取数据；允许合法的外部用户以指定的权限访问规定的网络资源。

防火墙是一个矛盾统一体，它既要限制信息的流通，又要保持信息的流通。因此，根据网络安全性总体需求，防火墙可遵循以下两种基本原则实现：

（1）一切未被允许的都是禁止的。根据这一原则，防火墙应封锁所有信息流，然后对希望提供的服务逐项开放。这种方法很安全，因为被允许的服务都是仔细挑选的，但限制了用户使用的便利性，用户不能随心所欲地使用网络服务。

（2）一切未被禁止的都是允许的。根据这一原则，防火墙应转发所有信息流，然后逐项屏蔽可能有害的服务。这种方法很灵活，可为用户提供更多服务，但安全性差一些。

由于以上两种防火墙原则在安全性和可使用性上各有侧重，很多防火墙系统在两者之间采取一定的折衷。

2. 防火墙的分类

根据不同的防范方式和侧重点，防火墙主要分成分组过滤型和代理服务型。它们在网络性能、安全性和应用透明性等方面各有利弊。

（1）分组过滤型。分组过滤型（Packet Filter）防火墙只是根据数据包的内容来判断是否允许数据包的传输。通常在网络层上通过对分组中的 IP 地址、TCP/UDP 端口号以及协议状态等字段的检查来决定是否转发一个分组。

分组过滤型防火墙的基本原理是：

1）根据网络安全策略，在防火墙中事先设置分组过滤规则。

2）依据分组过滤规则，对经过防火墙的分组流进行检查，通常需要检查下列分组字段：

- 源 IP 地址和目的 IP 地址；
- 协议类型：TCP 分组、UDP 分组还是 ICMP 分组；
- 源 TCP 端口和目的 TCP 端口；
- 源 UDP 端口和目的 UDP 端口；
- ICMP 消息类型；
- 分组的输出接口。

3）分组过滤规则一定要按顺序排列。当一个分组到达时，按规则的排列顺序依次运用每个规则对分组进行检查。一旦分组与一个规则相匹配，则不再向下检查其他的规则。

4）如果一个分组与一个拒绝转发的规则相匹配，则该分组将被禁止通过。

5）如果一个分组与一个允许转发的规则相匹配，则该分组将被允许通过。

6）如果一个分组不与任何规则相匹配，则该分组将被禁止通过，这是遵循“一切未被允许的都是禁止的”原则。

分组过滤型防火墙速度快且对用户透明，比较容易建立且价格便宜，但也有一些潜在

的弱点，如分组过滤规则叙述起来很复杂，过滤规则集太绕弯，漏洞很难检测到。

（2）代理服务器。代理服务型防火墙充当客户和服务器程序之间的中介。客户程序要访问服务器必须通过代理服务器进行中转，而不允许它们之间直接建立连接进行通信。代理服务器打开一个到服务器的连接，发出请求，并继续在服务器和客户之间传输数据。

克服分组过滤弱点的一个通用方法是使用一个应用程序级网关。应用程序级网关是一个运行 proxy 服务的主机，用于过滤 telnet 和 FTP 这类服务连接，它可以和一个分组过滤路由器组合起来提供更好的安全性和灵活性。

在这种安全体系结构中，Intranet 通过代理服务器向 Internet 开放某些服务，如 HTTP、Telnet 及 FTP 等。当外部客户访问这些服务时，所连接的是代理服务器，而不是实际的服务器，但外部客户感觉是实际的服务器。代理服务器根据请求者的身份、服务类型、服务内容、域名范围、登录时间等因素来确定是否接受客户请求。如果接受客户请求，则代理服务器代替该客户向实际服务器发出请求，实际服务器返回的结果再由代理服务器传送给外部客户。

如果不接受客户请求，则代理服务器直接向该客户发出拒绝服务的信息。在这种防火墙中，每一种 Internet 应用都需要有相应的代理软件，如 HTTP 代理、FTP 代理及 Telnet 代理等，并且提供认证、审计及日志等功能。

通常，运行代理服务器软件的主机称为堡垒主机（Bastion Host）。在体系结构上，代理服务型防火墙可采用 3 种结构来实现。

1）双穴主机网关（Dual Home Gateway）。这种结构是在堡垒主机上插入两块网卡，由堡垒主机充当内部网与 Internet 之间的路由器，并在其上运行代理服务器软件，受保护的内部网与 Internet 之间不能直接建立连接，必须通过堡垒主机才能通信。外部用户只能看到堡垒主机，而看不到内部网的实际服务器和其他资源。受保护网络的所有开放服务必须通过堡垒主机上的代理服务软件来实施。这种结构的代理服务器安全性比较脆弱，一旦防火墙失效，堡垒主机将变成一个丧失寻径功能的路由器，有经验的入侵者可以恢复它的寻径能力，进而攻入内部网。

2）屏蔽主机网关（Screened Host Gateway）。这种结构是在内部网与 Internet 之间设置一个带屏蔽功能的路由器，堡垒主机连接在内部网上。屏蔽路由器基于下列规则过滤分组：堡垒主机是内部网惟一的系统，允许外部客户与堡垒主机建立连接，并且只能通过与堡垒主机建立连接来访问内部网提供的服务。同样，堡垒主机要运行代理服务器软件。这种结构的代理服务器设有两道安全屏障：屏蔽路由器和堡垒主机，并且这两道安全屏障可以相互协调和补充，因此，具有较高的安全性，并且比较灵活。

3）屏蔽子网网关（Screened Subnet Gateway）。这种结构是在内部网与 Internet 之间设置一个独立的子网，在内部网与子网之间和子网与 Internet 之间都要设置一个屏蔽路由器，堡垒主机连接在子网上，并运行代理服务器软件。堡垒主机是惟一的内部网和 Internet 能访问的系统，但要受到屏蔽路由器过滤规则的限制。这种结构的代理服务器设有 3 道安全屏障：两个屏蔽路由器和堡垒主机。外部非法用户要入侵内部网必须攻破两个屏蔽路由器和堡垒主机，这显然是相当困难的。因此，它具有更高的安全性，比较适合保护大型网

络，但成本也比较高。

在实际应用中，代理服务型防火墙有两种类型：一种是应用级网关型，它工作在应用层上，对每一种应用，都设有一个代理，前面所讲的代理服务器均属于这类防火墙；另一种是电路级网关型，它工作在传输层上，根据客户的 TCP 连接请求，重新建立一个允许通过防火墙的 TCP 连接（也称连接重定向）来提供代理服务，而不是针对应用程序。两者相比，前者的安全性好，而灵活性和透明性不如后者。

此外，在代理服务型防火墙中还采用一种叫做网络地址翻译器（NAT）的代理技术，NAT 允许在内部网络中使用私有 IP 地址，而私有 IP 地址在 Internet 中是不可见的，可见的只是代理服务器的 IP 地址。这就意味着内部网络中可以使用任何 IP 地址，不一定是 Internet 中惟一的地址。

分组过滤型和代理服务型防火墙技术可单独使用，也可组合在一起使用，它们在安全性、透明性和价格方面各有利弊。总体说来，分组过滤型防火墙的透明性好，价格便宜，但安全性较弱，而代理服务型防火墙在安全性比较强，但在透明性和价格上略逊一筹。

防火墙对来往的分组进行检查需要花费一定的处理时间，在无形中增加了分组的传输延迟，因此，在某种程度上，防火墙可以看成是一种网络瓶颈。

7.4.4 Linux 常用的网络命令

计算机网络的主要优点是能够实现资源和信息的共享，并且用户可以远程访问信息。Linux 提供了一组强有力的网络命令来为用户服务，这些工具能够帮助用户登录到远程计算机上，传输文件和执行远程命令等。

下面介绍几个常用的有关网络操作的命令。

1. ftp 命令

ftp 命令是标准文件传输协议的用户接口。ftp 是 TCP/IP 网络的计算机之间传输文件的简单有效的方法，它允许用户传输 ASCII 文件和二进制文件。

ftp 命令的功能是在本地机和远程机之间传送文件。该命令的一般格式如下：

$ftp 主机名/IP

其中，“主机名/IP”是所要连接的远程机的主机名或 IP 地址。在命令行中，主机名属于选项，如果指定主机名，ftp 将试图与远程机的 ftp 服务程序进行连接；如果没有指定主机名，ftp 将给出提示符，等待用户输入命令：

$ ftp

ftp >

此时，在 ftp>提示符后面输入 open 命令加主机名或 IP 地址，将试图连接指定的主机。

不管使用哪一种方法，如果连接成功，都需要在远程机上登录。如果用户在远程机上有帐号，可以通过 ftp 使用这一帐号并提供口令。远程机上的用户帐号的读写权限决定该用户在远程机上能下载什么文件和将上载文件放到哪个目录中。

如果没有远程机的专用登录帐号，许多 ftp 站点设有可以使用的特殊帐号。这个帐号的登录名为 anonymous（也称为匿名 ftp），当使用这一帐号时，要求输入 email 地址作为

口令。

如果远程系统提供匿名 ftp 服务，用户使用这项服务可以登录到特殊的、供公开使用的目录，一般专门提供 pub 目录和 incoming 目录。pub 目录包含该站点供公众使用的所有文件，incoming 目录存放上载到该站点的文件。

一旦用户使用 ftp 在远程站点上登录成功，将得到“ftp>”提示符，可以自由使用 ftp 提供的命令，可以用 help 命令取得可供使用的命令清单，也可以在 help 命令后指定具体的命令名称，获得这条命令的说明。

2. telnet 命令

用户使用 telnet 命令进行远程登录。该命令允许用户使用 telnet 协议在远程计算机之间进行通信，用户可以通过网络在远程计算机上登录，就像登录到本地机上执行命令一样。

为了通过 telnet 登录到远程计算机上，必须知道远程机上的合法用户名和口令。虽然有些系统确实为远程用户提供登录功能，但出于对安全的考虑，要限制来宾的操作权限，因此，这种情况下能使用的功能是很少的。当允许远程用户登录时，系统通常把这些用户放在一个受限制的 shell 中，以防系统被怀有恶意或不小心的用户破坏。

用户还可以使用 telnet 从远程站点登录到自己的计算机上，检查电子邮件，编辑文件或运行程序，就像在本地登录一样。但是，只能使用基于终端的环境而不是 X Windows 环境，telnet 只为普通终端提供终端仿真，而不支持 X Windows 等图形环境。

telnet 命令的一般格式为：

telnet 主机名/IP

其中“主机名/IP”是要连接的远程机的主机名或 IP 地址。如果这一命令执行成功，将从远程机上得到“login：”提示符。

使用 telnet 命令登录的过程如下：

$ telnet 主机名/IP

启动 telnet 会话。

一旦 telnet 成功地连接到远程系统上，就显示登录信息并提示用户输入用户名和口令。如果用户名和口令输入正确，就能成功登录并在远程系统上工作。

在 telnet 提示符后面可以输入很多命令，用来控制 telnet 会话过程。

3. rlogin 命令

rlogin 是 remote login（远程登录）的缩写。该命令与 telnet 命令很相似，允许用户启动远程系统上的交互命令会话。rlogin 的一般格式是：

rlogin [-8EKLdx] [-e char] [-k realm] [- l username] host

一般最常用的格式是：

rlogin host

该命令中各选项的含义如下：

-8 始终允许 8 位输入数据通道。该选项允许发送格式化的 ANSI 字符和其他的特殊代码。如果不用这个选项，除非远端的终止和启动字符，否则将去掉奇偶校验位。

-E 停止把任何字符当作转义字符。当和-8 选项一起使用时，提供一个完全的透明连接。

-K 关闭所有的 Kerberos 确认。只有与使用 Kerberos 确认协议的主机连接时才使用这个选项。

-L 允许 rlogin 会话在 litout 模式中运行。要了解更多信息，请查阅 tty 联机帮助。

-d 打开与远程主机进行通信的 TCP sockets 的 socket 调试。要了解更多信息，请查阅 setsockopt 的联机帮助。

-e 为 rlogin 会话设置转义字符，默认的转义字符是“~”，用户可以指定一个文字字符或一个\\nnn 形式的八进制数。

-k 请求 rlogin 获得在指定区域内的远程主机的 Kerberos 许可，而不是获得由 krb_realmofhost（3）确定的远程主机区域内的远程主机的 Kerberos 许可。

-x 为所有通过 rlogin 会话传送的数据打开 DES 加密，这会影响响应时间和 CPU 利用率，但是可以提高安全性。

4. rsh 命令

rsh 是 remote shell（远程 shell）的缩写。该命令在指定的远程主机上启动一个 shell，并执行用户在 rsh 命令行中指定的命令。如果用户没有给出要执行的命令，rsh 就用 rlogin 命令使用户登录到远程机上。

rsh 命令的一般格式是：

rsh [-Kdnx] [-k realm] [-l username] host [command]

常用的格式是：

rsh host [command]

command 可以是从 shell 提示符下键入的任何 Linux 命令。

rsh 命令中各选项的含义如下：

-K 关闭所有的 Kerbero 确认。该选项只在与使用 Kerbero 确认的主机连接时才使用。

-d 打开与远程主机进行通信的 TCP sockets 的 socket 调试。要了解更多的信息，请查阅 setsockopt 的联机帮助。

-k 请求 rsh 获得在指定区域内的远程主机的 Kerberos 许可，而不是获得由 krb_relmofhost（3）确定的远程主机区域内的远程主机的 Kerberos 许可。

-l 缺省情况下，远程用户名与本地用户名相同。本选项允许指定远程用户名，如果指定了远程用户名，则使用 Kerberos 确认，与在 rlogin 命令中一样。

-n 重定向来自特殊设备/dev/null 的输入。

-x 为传送的所有数据打开 DES 加密，这会影响响应时间和 CPU 利用率，但可以提高安全性。

Linux 把标准输入放入 rsh 命令中，并把它复制到要远程执行的命令的标准输入中，并把远程命令的标准输出复制到 rsh 的标准输出中，还把远程标准错误复制到本地标准错误文件中。任何退出、中止和中断信号都被送到远程命令中。当远程命令终止了，rsh 也就终止了。

5. rcp 命令

rcp 是 remote file copy（远程文件复制）的缩写。该命令用于在计算机之间复制文件。

rcp 命令有两种格式。第一种格式用于文件到文件的复制；第二种格式用于把文件或目录复制到另一个目录中。rcp 命令的一般格式是：

rcp [-px] [-k realm] file1 file2

rcp [-px] [-r] [-k realm] file directory

每个文件或目录参数既可以是远程文件名也可以是本地文件名。远程文件名具有如下形式：

rname@rhost：path

其中，rname 是远程用户名，rhost 是远程计算机名，path 是这个文件的路径。

rcp 命令的各选项含义如下：

-r 递归地把源目录中的所有内容复制到目的目录中。要使用这个选项，目的地必须是一个目录。

-p 试图保留源文件的修改时间和模式，忽略 umask。

-k 请求 rcp 获得指定区域内的远程主机的 Kerberos 许可，而不是获得由 krb_relmofhost（3）确定的远程主机区域内的远程主机的 Kerberos 许可。

-x 为传送的所有数据打开 DES 加密，这会影响响应时间和 CPU 利用率，但是可以提高安全性。

如果在文件名中指定的路径不是完整的路径名，那么这个路径被解释为相对远程机上同名用户的主目录。如果没有给出远程用户名，就使用当前用户名。如果远程机上的路径包含特殊 shell 字符，需要用反斜线（\\）、双引号（“”）或单引号（‘’）括起来，使所有 shell 字符都能被远程解释。

需要说明的是，rcp 不提示输入口令，它通过 rsh 命令来执行复制。

本章小结

TCP/IP 协议是 Linux 使用的主要协议，是面向连接的传输控制协议。IP 协议工作在网络层，负责数据报在计算机之间寻址，决定数据传送到哪里及出现网络故障时如何更换路由。TCP 协议提供无连接的传输控制服务。

在配置 Linux 网络中，主要讲了基本的网络配置文件及如何手工配置 Linux 网络。

DNS 服务器提供了名字转换功能，是由解析器查询的。解析器是专门用于在域名服务器中获取 IP 地址的程序。要想在系统上使用域名服务器，必须配置解析器。本地的解析器是通过文件/etc/host.conf 和/etc/resolv.conf 配置的。

Web 服务是 Internet 服务器最基本的服务，Linux 发行版中包含的 Apache 软件是性能优良的 Web 服务器，也是 Internet 上最流行的 Web 服务器，由于它时刻都经历着无数使用者的测试，所以现行 Apache 的默认选项已经十分适合大家了，只需要更改其中几个与当前应用环境紧密相关的选项就可以达到你的目的。

sendmail 有很强的适用性，并且是在不断更新的。所以现在提供的 sendmail 的配置方案已经经过实践检验，不需要管理员做太大改动，就可以实现强大的 sendmail 功能。

Linux 是一个优秀的网络操作系统，可与多种网络集成。Linux 系统的稳定性、可靠性受到了广大用户的欢迎，在中小型网或公司、部门、单位等内部网（Intranet）上，Linux 常充当有效而强劲的文件和打印服务器，让 Windows 客户机共享 Linux 系统中的文件。这种 Linux 与 Windows 网络集成是通过 samba 实现的。看了本文关于 Samba 的介绍后，读者可能有一个切身的感觉，就是 Samba 的配置选项太多了，但只需更改其中不多的部分就足以实现强大的文件、打印共享了。

在 Linux 网络安全一节中介绍了文件系统的安全、用户口令的安全、防火墙技术及常用的 Linux 网络命令。

习 题 七

7-1 讲述计算机网络的分层体系结构。

7-2 简述 IP 地址的分类。

7-3 在用户系统的/etc 目录下查看 hosts 和 host.conf 文件，执行添加主机名等简单操作，查看结果。

7-4 查看 named 的配置文件。

7-5 阅读 httpd.conf 文件。

7-6 阅读 sendmail 的帮助文件。

7-7 适当修改 samba 配置，查看系统有何不同。

7-8 简述 DNS 配置过程。

7-9 简述防火墙的定义及分类。

7-10 简述配置 linux 网络的过程。

第 8 章　现代操作系统实例

本章学习目标

前面各章在介绍操作系统原理的基础上，以 Linux 系统为实例做了说明。本章将对目前其他一些常见的操作系统从结构及功能上进行系统介绍。通过本章的学习，读者应掌握以下内容：

- UNIX 系统的内核管理方法
- Windows 系统的内核管理方法
- 分布式操作系统特性及进程管理方式

目前最常用的操作系统是 Windows、UNIX 和 Linux。其中，Windows 系统包括 Windows 3.x、Windows 95、Windows 98、Windows 2000 / NT 及 Windows Me、Windows XP 等。UNIX 常用的变种有 SUN 公司的 Solaris、IBM 公司的 AIX、惠普公司的 HP UX 等。其他比较常用的操作系统还有 Mac OS、NetWare、OS/2 等。

8.1　UNIX 操作系统

8.1.1　UNIX 操作系统的发展

UNIX 是目前最流行的操作系统之一，它为用户提供了一个简洁、高效、灵活的运行环境。与其他操作系统相比，UNIX 操作系统功能更强，应用更加广泛，并且具有更高的安全可靠性。因此，UNIX 实际上已经成为工业计算机系统的标准。

UNIX 于 1969 年在美国的电报电话公司（AT&T）贝尔实验室（Bell Labs）诞生。1970 年，UNIX 开始在 PDP-7 机上运行，当时只能支持两个人同时使用。最初的 UNIX 系统是用汇编语言编写的。1973 年，Ritchie 又用 C 语言重写了 UNIX。1976 年正式公开发表了 UNIV V版本，并开始向美国各大学及研究机构颁发使用许可证，公开了源代码。

随着微型计算机技术的巨大发展，1980 年，为满足微型计算机用户使用 UNIX 的需要，Microsoft 公司根据微型计算机的特点对 UNIX 的第七版进行了修改和扩充，并改名为 XENIX，随后微软公司将 XENIX 的版权授于 SCO 公司，并由 SCO 公司负责微机的开发和利用。

8.1.2　UNIX 操作系统的特点

UNIX 操作系统取得巨大成功的根本原因是它本身的优越性能和特点。UNIX 是一个

分时、多用户、多任务的操作系统，它具有一般操作系统所具有的特点，如生成和管理文件系统，内存和外设管理。除此之外，与其他操作系统相比，它还具有以下特点。

1. 多用户多任务

UNIX 是一个多用户、多任务的操作系统，允许系统中的每位用户运行不同的程序，就像拥有一台单独使用的计算机。多任务是指 UNIX 支持在同一台主机上运行多道程序。UNIX 通过一个分时处理程序来实现多任务。

2. 可移植性

许多操作系统是由汇编语言写成的，对于不同硬件设备很难移植，而 UNIX 系统是由 C 语言编写的，对它进行移植可以不考虑硬件设备的影响，所以能很方便地移植。

3. 树形文件系统结构

UNIX 的树形目录结构称为有向非循环图结构，根目录为树根，其他目录是分枝，但 UNIX 的文件系统是可以挂接的。这种结构不但易于管理，而且有利于系统的安全和保密。

4. I/O 重定向和管道技术

I/O 重定向技术是指程序在何处得到结果以及将结果送往何处，利用命令级接口，通常将执行的结果显示到屏幕上，但是通过 I/O 重定向技术可以让 UNIX 命令的结果输出到指定的文件。

5. 丰富的实用程序

UNIX 提供了大量的实用程序供用户使用，如 vi、ed 等。通过这些实用程序，用户能轻易地完成特定的工作。

6. 电子邮件

UNIX 具有很强的电子邮件功能，要将邮件发送给同一系统下的另一个用户，只需知道该用户的用户名即可；要将邮件发送给另一个 UNIX 系统下的用户，只要知道该用户的网络地址即可。

8.1.3 UNIX 系统的内核结构

UNIX 操作系统是层次结构的模型。整个系统可分为 4 个层次，最底层是硬件，作为整个系统的基础；次底层是操作系统的核心，包括了操作系统的四大资源管理功能；再往上是操作系统与用户的接口 Shell 及编译程序等；最高层是应用程序层，如图 8-1 所示。

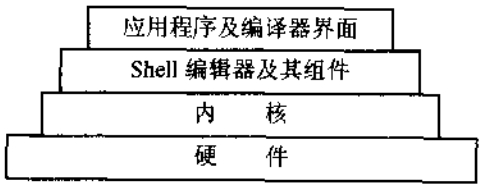


图 8-1　UNIX 系统层次结构模型

用户程序可以通过高级语言的系统调用或低级语言的直接系统调用进入核心。系统调用接口是内核与用户的界面，并允许高层软件访问特定的内核功能；操作系统中包含的原

语操作直接对硬件起作用。在这两个界面之间，系统被分为两部分，一部分主要从事进程控制，另一部分进行文件管理和 I/O 控制。进程控制子系统负责内存管理、进程调度以及进程同步和线程间的通信；文件管理子系统包括控制对文件的访问、分配文件的存储器空间等文件操作功能。文件子系统通过一个缓冲机制同设备驱动部分相互作用，也可以在无缓冲机制参与的条件下与字符设备相互作用。设备管理、进程管理及存储管理通过硬件控制接口与硬件相互作用。如图 8-2 所示为 UNIX 系统的内核模型。

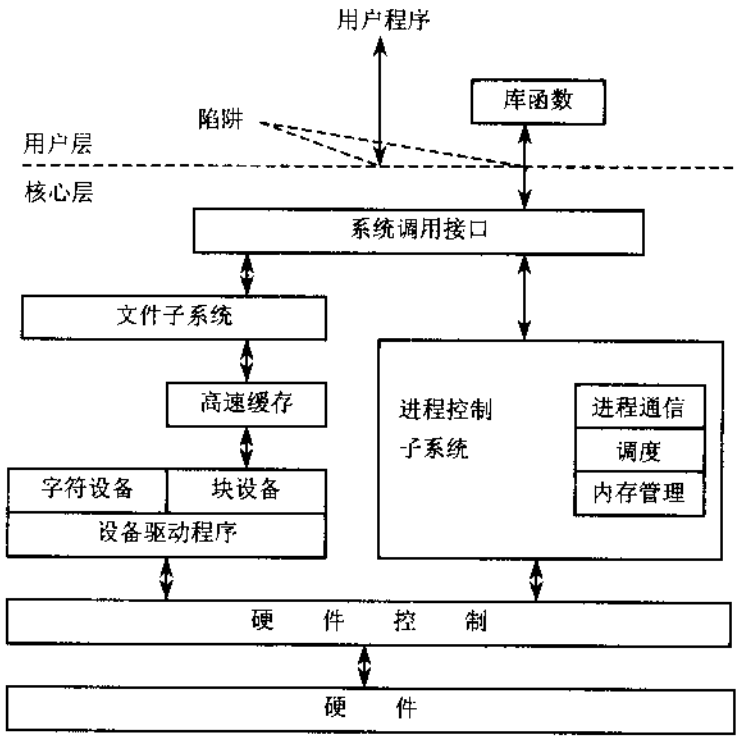


图 8-2　UNIX 系统的内核模型

8.1.4　UNIX 系统的进程管理

1. UNIX 进程的结构

在 UNIX 系统中，进程是进程映像的执行过程。进程的结构也称为进程实体，由用户级上下文、寄存器上下文和系统级上下文 3 部分构成。

（1）用户级上下文。用户级上下文主要成分是用户程序，包括正文段即代码（text）、数据段（data）两部分。

（2）寄存器上下文。寄存器上下文主要是由 CPU 中的一些寄存器内容组成的。

（3）系统级上下文。系统级上下文包括操作系统为管理该进程所用的信息，可分为静态和动态两部分，包括：

proc 结构：该结构常驻内存，内容包括经常需要访问的往息，如进程标识符、进程状

态等。

user 结构：该结构暂驻内存，进程处于执行状态时调入内存，它包含了进程的一些私有信息，如进程表项指针、有效用户标识符等各种资源表格。

进程区表：从虚拟地址到物理地址的映射。

核心栈：核心态执行时过程调用的栈结构。

若干寄存器级上下文。

2. 进程调度算法

UNIX 系统是分时系统，它的进程调度采用动态优先数轮转调度算法。调度程序进行调度时，首先从“内存就绪”或“被抢占”状态的进程中选择一个优先级最高的进程。UNIX 系统的进程优先级可用相应的优先数来表示，优先数越小，优先级别越高。例如，对换进程的优先数是 0，而等待磁盘 I/O 进程的优先数是 20。

UNIX S_5 中进程的优先级分为用户优先级类和核心优先级类。核心用两种方式改变进程的优先级：对核心态进程设置优先数；对用户态进程计算优先数。

（1）核心态进程因等待某一事件而调用 sleep 程序睡眠时，核心根据该进程睡眠的原因，为它设置一个确定的优先数。设置优先数的原则不是取决于该进程运行时间长短（I/O 型或计算型），而是取决于它睡眠的原因。

（2）系统调用执行结束，进程由核心态返回用户态，以及进程正在用户态下运行时，核心通过计算方式来调整其优先数。一个运行的进程在其时间片内会多次被时钟中断。每次中断时，时钟处理程序就增加该进程当前使用 CPU 的时间值。每到一秒，时钟处理程序就根据一个函数调整各个进程当前使用 CPU 的时间值：

decay(p_cpu)=p_cpu/2

其中，p_cpu 为当前进程使用处理机的时间值。此外，时钟处理程序也在为内存就绪状态的进程重新计算优先数：

进程优先数＝最近使用 CPU 的时间 / 2＋基本用户优先数

基本用户优先数也称分界优先数。在 UNIX S_5 中，分界优先数为 60，它是用户态进程的最小优先数。

为重新计算优先数，优先级为用户级别的进程会从一个优先级队列移到另一个队列，此时核心并不改变核心态进程的优先数，也不允许具有用户级别优先级的进程穿越分界优先级而进入核心级别优先级范围中。当然，用户态进程可以通过执行系统调用，并且睡眠而获得核心态优先级。

3. 进程状态及转换

UNIX S_5 进程的状态存放在进程的 proc 结构中，这些进程的状态共有 9 个。

（1）用户态执行。表示进程在用户态下执行的状态。

（2）核心态执行。表示进程在核心态下执行的状态。进程是在用户态下还是在核心态下执行主要取决于处理机状态字 PSW。

（3）内存中睡眠。进程正在内存中处于睡眠状态。如果进程所执行的系统调用涉及到 I/O 操作，而进程又必须等待它的完成，则进程将进入内存中睡眠状态。

（4）睡眠且换出。当内存紧张时，在内存中睡眠的进程首先被核心换出到外存，以腾出相应的内存空间，此时，进程为睡眠且换出状态。

（5）内存中就绪。进程处于就绪状态，只要核心程序调度到它，就可以执行。

（6）就绪且换出。虽然进程处于就绪状态，但交换程序已将其唤出内存，只有等待交换程序把它换入内存后，核心才能调度它去执行。

（7）僵死。进程执行了 exit 后，进程已不存在，但它留下一些含有状态码和计时系统信息的记录，供父进程收集，这种状态称为僵死状态。

（8）被剥夺状态。进程正从核心态返回用户态，但核心调度程序抢先剥夺了该进程的处理机，以调度其他进程的执行，最后，当该进程再次被调度时返回用户态。

（9）创建。当进程刚被创建时，正在进行资源分配，因此，它既不是就绪状态，也不是睡眠状态，这个状态被称为进程的初始状态，或称创建状态。

如图 8-3 所示，反映了一个进程从被创建到被撤消的整个生命过程中的变化情况。各状态之间的转换有些是通过系统原语或核心函数完成的，如唤醒或调度等，有些则由外部事件的发生而导致。

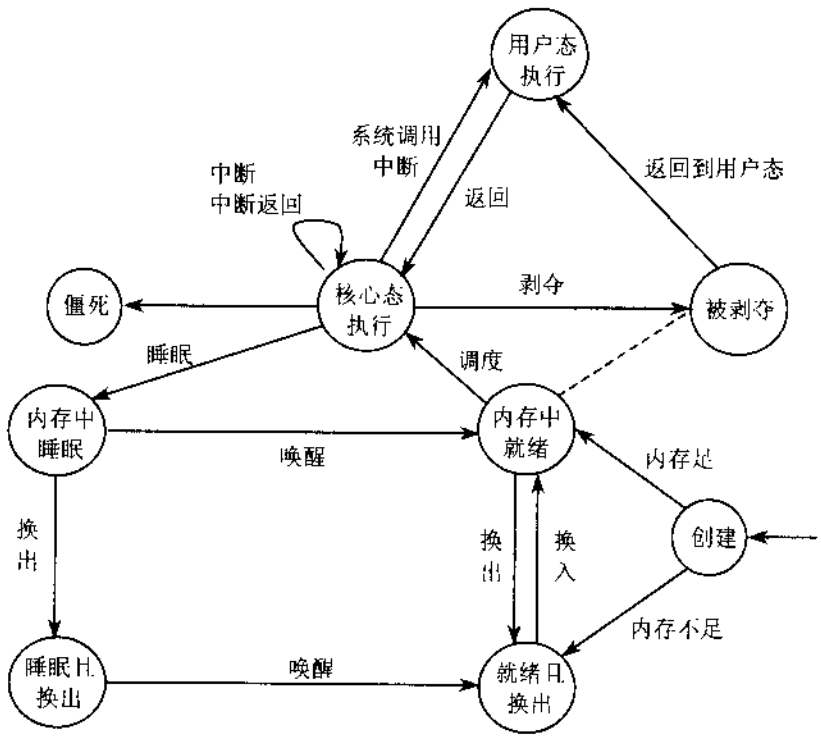


图 8-3　UNIX 系统的进程状态转换

首先，当父进程执行系统调用 fork 时，被创建进程进入创建状态。在该状态下，核心为该进程分配 user 结构区以及必要的内存工作集。内存管理分配程序如果能为该进程分配足够的内存，则进程的状态发生变化，由创建状态进入内存就绪状态。此时，由于该进程

已分得 user 结构区、各种页表、堆栈、部分正文段和数据段所用的内存空间，因此，该进程可以经调度程序选中后占用 CPU 并执行。

如果内存分配程序不能为该进程分配足够的内存，则该进程的进程上下文被放入外存交换系统中，进程状态则由创建状态进入外存就绪状态。处于该状态的进程，只有在交换程序将其进程的上下文换入内存，进入内存就绪状态后，才有可能被调度执行。

当进程进入内存就绪状态后，进程调度程序将可能选择该进程执行。这时，该进程在核心态下执行，以装配该进程的进程上下文。在这个状态下，该进程完成创建部分的工作。

当进程完成 fork 系统调用后，可能返回用户态下执行用户程序，这时进入用户态执行状态。

在进程完成系统调用后返回用户态之前，根据 UNIX S_5 的调度机制，此时若有优先级高于当前进程的进程进入内存中就绪态，则系统将抢占处理机，此时，该进程进入被剥夺状态。进程进入被剥夺状态后，在下一次进程调度时，系统分配给它 CPU 才能返回用户态。

进程在核心态执行时，因为等待某事件发生，调用 sleep 原语进入睡眠状态。处于睡眠状态的进程因为内存的限制，将在睡眠一段时间后，被交换程序换出内存而进入外存睡眠状态，直到所等待的事件发生并将它唤醒，而进入内存中睡眠状态。

进程完成它所要求的任务后，将使用系统调用 exit，从而使进程进入僵死状态，释放资源。

4. UNIX 系统中进程的家族关系

UNIX 系统中众多进程存在家族关系：由父进程创建子进程，子进程再创建它的子进程，从而构成了树形的进程家族图。UNIX 系统的进程家族关系如图 8-4 所示，图中节点表示进程。

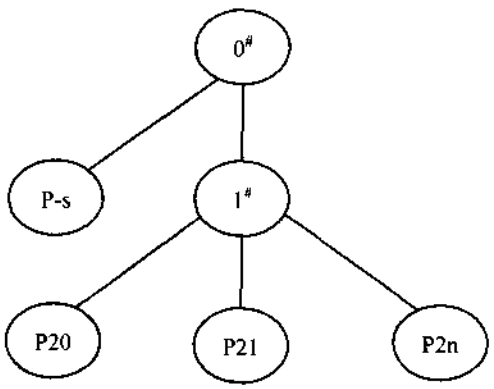


图 8-4 UNIX 进程的家族关系

UNIX 的内核中设置了一个 0#进程，它是惟一一个在系统引导时被创建的进程。系统初启时，由 0#进程再创建 1#进程及其他核心进程，然后 1#进程又为每个终端创建命令解释进程，用户输入命令后又创建若干进程，这样便形成了一棵进程树。以后，0#进程作为系统的对换及调度进程，1#进程成为系统始祖进程，同时又创建其他进程。系统中除 0#进程

以外，所有其他进程都是由 fork 创建的。

8.1.5　UNIX 系统的内存管理

UNIX 系统采用求请调页存储管理方式，支持内外存的对换功能。内存空间的分配和回收均以页为单位进行。当进程运行时，不必将整个进程映像加载内存，只需将当前要用的页面装入内存即可，这样，用户可以得到更大的逻辑地址空间。下面简单介绍 UNIX 系统的交换技术和请求分页技术。

1. 交换

在 UNIX 系统中，由于多个进程并发执行，内存资源十分有限。为此引入了交换策略，将内存中处于睡眠状态的某些进程调到外存交换区中，而将交换区中的就绪进程重新调入内存。为了实现这种策略，系统内核应具有交换空间的管理、进程换出和进程换入 3 个功能。

（1）交换空间的管理。由于进程在交换区驻留的时间较短，交换操作较为频繁，因此，在交换空间的管理中，速度显得较为重要。为此，内核较少考虑存储空间的碎片问题。分配交换空间使用的数据结构驻留在内存的交换映射表。

（2）进程的换出。当内核需要内存空间时，可把一个进程换到交换区中，当内核决定一个进程适合换出时，先将该进程每个区的引用计数减 1，然后选出值为 0 的进程换出。另外，内核还需调用 malloc 过程来申请交换空间，并对该进程加锁。

（3）进程的换入。引用计数为 0 的进程就是交换进程。当睡眠进程被唤醒去执行换入操作时，便去查看进程表中所在进程的状态，从中找出“就绪且换出”状态的进程，把其中换出时间最久的进程作为换入进程，再根据该进程的大小调用 malloc 过程，为其申请内存。当申请内存成功时，直接将进程换入，否则需先将内存中的某些进程换出，腾出足够的内存空间后再将进程换入。

2. 请求分页

UNIX 系统为实现请求分页的功能配置了 4 种数据结构。

（1）页表。页表用于将逻辑页号映射为物理块号。页表项包括页的内存地址、读/写或执行的保护位和一些附加的信息位，如有效位、访问位、修改位、复制写位和年龄位等。

（2）磁盘块描述字。每一个页表项对应一个磁盘块描述字，其中记录了进程不同时刻所在的各虚拟页的磁盘备份的块号。当进程在运行中发现缺页时，可通过查找该页而找到所需调入的页面位置。

（3）页面数据表。页面数据表用于描述每个物理页。通过页号进行索引，数据表的内容包含页面状态、访问该页面的进程数目、逻辑设备和该页所在的盘块号以及一些指针，这些指针指向空闲页面链表和页面散列队列中的其他页框数据表项。

（4）交换使用表。交换使用表描述设备上每一页的使用情况，每个交换设备上的页面在交换使用表中都占用一项，该项表明有多少表指向交换设备上的一个页面。

3. 换页进程

换页进程是核心进程，该进程的主要任务是增加内存中所有的有效页年龄，并将内存中长期不用的页面换出。

内存中的页面有可换出和不可换出两种状态。可换出页面年龄已到了规定的值。一个页面可计数的最大年龄取决于它的硬件设施。如对于只设置了两位的年龄域，其年龄只能取值 0、1、2、3。每当进程访问了某个页面时，便将该页面的年龄置为 0。因此，当一个页面被连续检查了 3 次，且中间未被访问过时，其年龄才可能增至 3，而成为被换出的页。

如果多个进程共享一个分区，则相应页面可被多个进程同时使用，只要页面被进程使用，它就应留在内存中，仅当不被任何进程使用时，才适于被换出。

4. 缺页

在 UNIX 系统中可能会出现两类缺页：有效缺页和保护性缺页。当出现缺页时，缺页处理程序可能要从盘上读一个页面到内存，并在 I/O 执行期间睡眠。

8.1.6 UNIX 系统的文件管理

UNIX 系统中的文件子系统既具有很强的功能，又具有灵活性。按文件的内部构造方式，UNIX 系统将文件分为普通文件、目录文件和特别文件（即设备文件）。UNIX 系统的目录结构为有向非循环图结构。

UNIX 的文件系统可分成基本文件系统和可装卸的子文件系统两部分。不论是基本文件系统，还是子文件系统，都有自己独立的结构。但是，基本文件系统是整个 UNIX 文件系统的基础，被称为根文件系统。系统一旦启动运行后，基本文件系统不能卸下，而子文件系统可以随时更换。这种结构使得文件系统易于扩充和更改。例如，用户可以把自己的文件组织在软盘上作为子文件系统，使用时将其插入，并用命令把它与基本文件系统装配在一起，这样，用户就可以像访问基本文件系统中的文件那样去访问子文件系统中的文件。使用完毕，可以卸载该子文件系统。

以下介绍 UNIX 系统文件管理的各部分功能。

1. 文件卷的组织结构

UNIX 系统中，文件是以块为单位存放在介质上的，存储介质可以是磁盘或磁带。通常把每个磁盘或磁带看作一个文件卷。如图 8-5 所示为 UNIX 系统的文件卷结构。

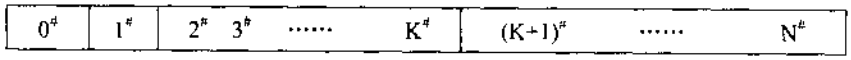


图 8-5 UNIX 系统的文件卷组织

其中，0#号块是系统的引导块或空闲，当该系统需引导时才有引导程序放在这里，其他一般文件系统都不使用引导块；1#块为超级块（也称为专用块），它既是文件系统的控制块，也是对空闲盘块和 I 节点等资源的管理表；从 2#块到第 K#（K 值由系统配置给定）块为索引节点区，用来存放该文件卷中所有文件的索引节点；从第(K+1)#块至第 N#块为文件区，用来存放系统中的所有文件。超级块包括以下内容：

（1）文件系统大小。包括 I 节点区所占盘块数和文件系统盘块总数。

（2）空闲盘块数目。指当前可被直接分配使用的盘块数。空闲盘块号栈和空闲盘块

号栈指针。

（3）空闲索引节点数目。

（4）空闲索引节点索引表。

（5）封锁标记（在维护空闲队列期间封锁）。

（6）专用块修改标记。

（7）其他信息。如总空闲块数、文件系统名称、文件系统状态等。

2. 文件的目录结构

UNIX 系统的目录结构采用了将文件名与文件描述信息分开的方法。文件目录由文件名和该文件的索引节点号构成。其中，文件名占 14 个字节，索引节点号（或索引节点指针）占 2 个字节。因此，1KB 的盘块中可以存放 64（1K/16）个目录项，这样就节省了系统查找及访问文件的时间。如图 8-6 所示为一个文件目录的实例。

文件名（14B）	磁盘索引节点号（2B）
F1.c	45
Myfile	38
F6.c	67
ABC	56
……	……

图 8-6　UNIX 系统的文件目录

一个文件的磁盘索引节点占 64 个字节，主要包括文件标识符、文件存取权限、文件物理地址、文件长度、文件连接系数、文件存取时间等一些文件的重要信息。

3. 文件的物理结构

UNIX 系统文件的物理结构采用混合索引方式，对分配给文件的磁盘块进行管理。在 UNIX 文件系统的索引节点中存在一项 i.addr[13]，用于存放该文件的磁盘块号。如图 8-7 所示为 UNIX 系统的混合索引文件结构。

（1）直接寻址。i.addr[0]~ i.addr[9]这 10 项用于直接存放该文件所占用的磁盘块号，文件的前 10 个磁盘块号依次放入其中。如果磁盘块的大小为 1KB，则当文件长度不大于 10K 时，操作系统可采用直接索引文件的方式对文件进行访问，直接从索引节点中找出该文件所在的磁盘块号，访问速度较快。

（2）一次间接寻址方式。i.addr[10]项中存放的磁盘块号所指向的磁盘块再用来存放下一级的磁盘块号。假设一个磁块大小为 1KB，每个磁盘块号占 4 个字节，则一个磁盘块可存放 256 个（1KB/4B）磁盘块号。这样，通过一次间接寻址，i.addr[10]项就引出了 256 个磁盘块号，所以，这一级可支持的文件长度为 256K。

（3）二次间接寻址方式。i.addr[11]存放的磁盘块号采用两级索引的方式，如果沿用以上的假设，则该项可引出 256^2 个磁盘块，可支持的文件长度为 256^2K。

（4）三次间接寻址方式。i.addr[12]存放的磁盘块号采用三级索引的方式，如果仍沿用

以上的假设，则该项可引出 256^3 个磁盘块，可支持的文件长度为 256^3K。

由以上分析可以看出，UNIX 系统对于长度较小的文件具有较快的读写速度，同时又具有支持大文件的功能。

4. 磁盘空间的管理方式

UNIX 系统对于空闲盘块的管理采用成组链接法。该方法把第一组中的所有空闲盘块号放入超级块的空闲盘块号栈中。

5. 系统为打开文件建立的数据结构

UNIX 系统打开文件的操作就是由操作系统在内存为文件建立相应的数据结构。系统为打开文件建立的数据结构有用户文件描述符表、文件表和内存索引节点。

3 个数据结构之间的关系如图 8-7 所示。

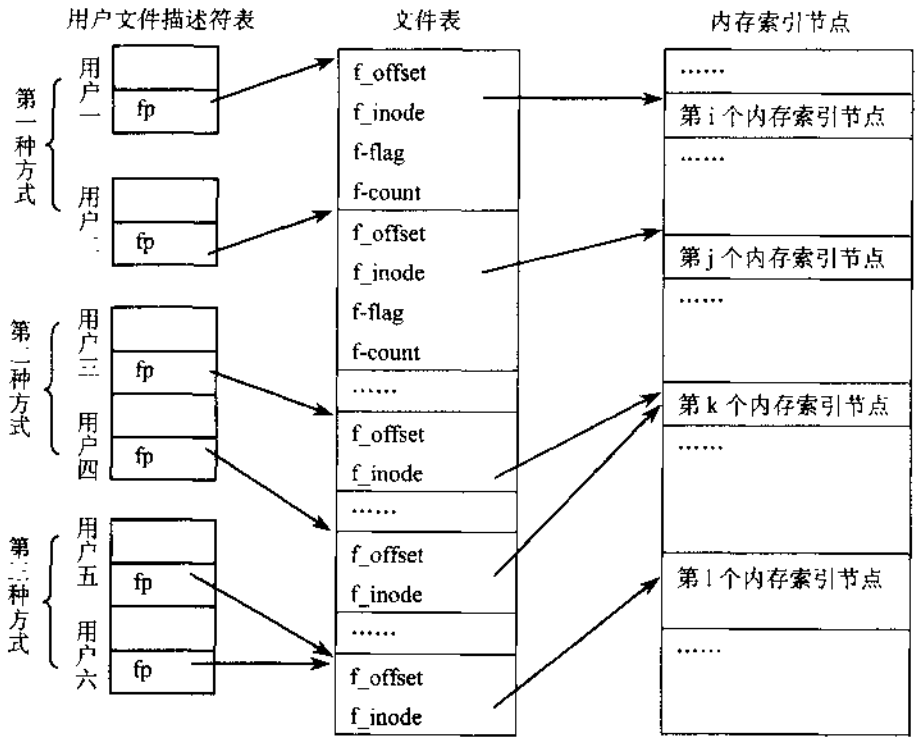


图 8-7 UNIX 打开文件在内存的 3 个数据结构及 3 种读写方式

（1）用户文件描述符表。为了方便用户和简化系统的处理过程，系统在每个进程的 U 区中都设置了一张用户文件描述符表。操作系统内核对打开请求进行检查后，便在该进程的用户文件描述符表中分配一个空项，取其在该表中的位移量作为文件描述符 fd 返回给用户。因为每个文件与它的文件描述符 fd 是一一对应的，所以当用户再访问该文件时，只需提供该文件的文件描述符 fd，系统便可找到相应文件的其他数据结构。

用户文件描述符表项的分配由 ufalloc 过程完成。

（2）文件表。系统为了方便对文件进行读/写访问，设置了一个确定读/写位置偏移量

的读/写指针 f_offset。在 UNIX 系统中引入文件表，将各用户对该文件的读/写指针放在该结构中。另外，还设置了文件引用计数 f_count，用于指示利用该文件表项的进程的数量；设置了指向内存索引节点的指针 f_inode 以及读/写标志 f_flag。

文件表的分配是由过程 falloc 完成的。

（3）对文件的读写方式。用户在访问文件可采用 3 种方式。多个用户读/写各自的文件；多个用户共享一个文件，但各自独立地对文件进行读/写；多个用户共享一个文件，且共享一个读/写指针。这样，对于上述 3 种读/写方式的访问要求都能够较好地满足。3 种读/写方式如图 8-7 所示。

8.1.7　UNIX 系统的设备管理

UNIX 系统把设备分为两类，即字符设备和块设备。UNIX S_5 将设备看作是一个特殊文件，因此，对设备的读/写操作和对文件的读写操作是一样的。这种方法为用户和进程提供了一个统一的接口。设备由文件系统进行管理。

UNIX 系统中有两种类型的 I/O，即缓冲 I/O 和无缓冲 I/O。缓冲 I/O 使用系统缓冲，而无缓冲 I/O 使用直接磁盘访问方式 DMA。在 DMA 方式中，传输直接在 I/O 模块和进程 I/O 区中发生。对于缓冲 I/O，使用系统缓冲 Cache 和字符队列。

1. 缓冲高速缓存

UNIX 系统中的缓冲 Cache 实际上是一个磁盘 Cache。对于磁盘的 I/O 操作通过缓冲 Cache 处理。缓冲 Cache 和用户进程空间之间的数据传输经常通过使用 DMA 发生，因为缓冲 Cache 和进程 I/O 都在内存中，DMA 机制利用这种方式来执行内存之间的复制。

当访问某个设备的一个物理块时，操作系统首先检查此块是否在缓冲区 Cache 中。为了提高检索速度，设备表列被组织成快表的形式。一个固定长度的快表中存放着指向缓冲 Cache 的指针。每一个对于设备号、块号的访问都被映射成快表中的一个特定入口。因此，对于所有（设备号、块号）访问都映射到同一快表中。块的替换使用的淘汰算法是 LRU 算法。当一个缓冲区分配给一个磁盘块后，只有当其他所有缓冲区最近都被访问过时，才可能将这一缓冲区分配给另一个块。

2. 字符队列

缓冲 Cache 可支持磁盘、磁带这样的块设备。另一种缓冲则适用于面向字符的设备，如终端、打印机等。字符队列可以由 I/O 设备写、处理器读或由处理器写、I/O 设备读。这两种情况都使用了第 2 章介绍的生产者—消费者模型，因此，字符队列只能读一次，一旦一个字符读过了，它就消失了。这和缓冲 Cache 不同，缓冲 Cache 可以读很多次，类似于前面介绍的读者—写者模型。

3. 无缓冲 I/O

无缓冲 I/O 是进程进行 I/O 最快的方法，它在设备和进程空间之间使用 DMA，进行无缓冲 I/O 的进程在主存中被锁起来，不能被换出。通过给高端内存加锁，减少了交换的机会，同时也降低了整个系统的性能，而且，I/O 设备固定于一个进程，在传输中不能为其他进程使用。

4. UNIX 设备

UNIX 系统本身可识别如下 5 种设备，即磁盘驱动器、磁带驱动器、终端、通信线、打印机。不同的 I/O 分别适用于不同的设备。如表 8-1 所示为设备和 I/O 之间的可用关系。

表 8-1 UNIX 中的设备 I/O

设备	无缓冲 I/O	缓冲 Cache	字符队列
磁盘驱动器	可用	可用	
磁带驱动器	可用	可用	
终端			可用
通信线			可用
打印机	可用		可用

5. 磁盘的读/写方式

（1）读方式。UNIX 系统有两种读方式，即一般读和提前读。

一般读方式：只把盘块中的信息读入缓冲区。

提前读方式：当一个进程要顺序地读一个文件所在的各个盘块时，会预见到所要读的下一个盘块，所以，读出一个指定的块时，可要求提前将下一个盘块中的信息读入缓冲区，这样可以提高系统的性能。该功能由 breada 过程完成。

（2）写方式。UNIX 系统有三种写方式，即一般写、异步写和延迟写。

一般写方式：把缓冲区中的数据写到磁盘上，且进程必须等待操作完成，由 bwrite 过程完成。

异步写方式：进程无须等待写操作完成便可返回，该过程由 bawrite 完成。

延迟写方式：该方式并不真正启动磁盘，而是在缓冲区首部置延迟写标志，然后释放该缓冲区，并将其链入空闲链表的末尾。当有进程申请到该缓冲区时，才将其真正写入磁盘。该方式由 bdwrite 过程完成。

8.2 Windows 2000 操作系统

8.2.1 Windows 2000 概述

Windows 操作系统家族是微软公司的核心产品之一，其产品包括 Windows 95、Windows 98、Windows Me 等桌面型操作系统以及 Windows NT、Windows 2000 服务器操作系统。从 1985 年开始，IBM 公司与微软公司合作开发商用多用户操作系统 OS/2，但这次合作并不十分融洽。1987 年 OS/2 推出后，微软公司开始计划建立自己的商用多用户操作系统。1988 年 10 月，微软公司开始组建开发新的操作系统，经过几百人四年多的工作，于 1993 年 5 月 24 日正式推出 Windows NT。在相继推出 Windows NT 1.0 到 Windows NT 4.0 等版本后，于 2000 年 2 月推出 Windows 2000（原来作为 Windows NT 5.0）。Windows 2000 操作系统结合了 Windows 98 和 Windows NT 4.0 很多优良的功能和性能，不仅增强了可靠性

和灵活性，增强了网络管理服务功能，降低了网络计算机系统的成本，并为运行企业级应用程序提供了最好的基础。Windows 2000 的下一个版本是 Windows XP 和 Windows2003。

Windows 2000 是一个商用多用户操作系统，其开发目标是开发工作站和服务器上的 32 位操作系统，以充分利用 32 位微处理器等硬件的新特性，并使它很容易适应将来的硬件变化，增加它的兼容性，同时，又不影响已有的应用程序的兼容性。

8.2.2　Windows 2000 中的进程和线程的特点

Windows 2000 操作系统对 CPU 的管理可体现为它的进程和线程模型。它具有以下特点：

（1）它是多任务（多进程）、多线程、对称多处理的。能够在任何可用的 CPU 上运行操作系统代码，并且可以在多个 CPU 上同时运行。

（2）由于采用了客户/服务器模型，Windows 2000 体现为多个服务器进程，每个服务器进程使用多个线程，用来处理同时从多个客户发来的请求。

（3）Windows 2000 的进程以及线程全部作为对象实现，并通过对象服务访问。

（4）进程管理程序在它所管理的子进程之间不具有父进程/子进程或其他关系。

（5）在该系统中，基本的调度实体是线程，它采用动态优先级多级调度队列算法，允许抢占 CPU，但核心代码不可被抢占。除了执行线程调度和中断的内核代码以外，当一个高优先级的线程需要引起注意时，所有的操作系统代码都可以被抢占。

（6）可以在进程间共享对象，以及具有灵活的内部进程通信能力。

（7）Windows NT 原设计可以在 Intel、MIPS 等多种 CPU 上运行，但出于市场因素的考虑，Windows 2000 只在 Intel 上运行。

进程和线程对象体由进程管理器管理，对象头由对象管理器管理，进程和线程对象在内核提供的内核进程对象和内核线程对象基础上实现。

8.2.3　Windows 2000 中进程的实现

1. 进程的构成

Windows 2000 中的进程由以下几部分组成：

（1）该进程所执行的程序。

（2）一个由该进程专用的地址空间。它是该进程可以使用的虚存地址的集合。

（3）系统资源。如文件、通信端口、信号量等。在程序执行时，由操作系统将这些资源分配给该进程。系统资源体现为对象表，每个进程都有一个对象表。

（4）至少有一个线程。进程的执行是通过线程实现的。一个进程可以有许多个线程，至少应有一个线程。

2. 进程的实现

在 Windows 2000 中，每个进程由一个进程对象结构表现，并有一个执行现场。进程执行现场包括进程的虚地址空间、该进程可见的资源集合、该进程的所有线程的集合。

每个进程都由一个进程环境块（PEB）描述，系统除内核外的其他部分是看不到 PEB 的。对于用户程序，看到的是执行体进程对象；而执行体以内的除内核外的其他部分，看

到的是内核对象，也被称作“内核进程”。内核进程对象是内核向执行体内的其他部分提供的，而执行体进程对象则是执行体向执行体外提供的。每个进程对象有一个内核进程对象与之对应，每个内核进程对象是其所对应的那个进程对象的子集。

8.2.4 Windows 2000 中线程的实现

1. 线程的构成

在 Windows 2000 中，一个线程由以下几部分构成：

（1）线程标识符。也称为客户 ID，它与线程是一一对应的。

（2）一组易变寄存器的内容，用来表示 CPU 的状态。易变是指线程切换后，这些寄存器的内容会被后来运行的线程覆盖，该线程要继续执行时，需恢复这些寄存器的内容。这些易变寄存器由哪些寄存器构成，是由 CPU 的结构决定的。通常包括程序计数器、CPU 状态、整数和浮点寄存器、同体系统结构相关的寄存器等。

（3）两个栈。一个是供线程在核心态下执行时使用，另一个是供线程在用户态下执行时使用。

（4）一个供该线程专用的存储区域，用来保护子系统、动态连接库等使用易变寄存器、栈（指针）和专用存储区域，称为该线程的现场。每个线程有一个线程现场，存储在线程对象体中。线程现场的组成是随 CPU 的改变而改变的。

2. 线程的状态

线程与进程一样，执行体线程对象是在内核线程对象基础上实现的。内核线程对象比内核进程对象更加复杂。它的内容主要包括：线程 CPU 族、执行线程所花的总时间、线程的基本优先级、线程的当前优先级、线程状态。Windows 2000 的线程状态共有以下 6 个：

（1）就绪（Ready）。

（2）备用（Standby）。处于备用状态的线程已被预定为某一特定 CPU 的下一个执行对象。系统中每个 CPU 只能有一个处于备用状态的线程。

（3）运行（Running）。

（4）等待（Waiting）。

（5）转换（Transition）。如果一个线程已准备好执行，但得不到它所需要的资源，此时线程进入转换状态。

（6）终止（Terminated）。线程执行完后便进入终止状态。一旦终止，线程对象是否被删除，由对象管理器决定。如果执行体有一指向线程对象的指针，它便可以将线程对象重新初始化，并再次使用它。

8.2.5 Windows 2000 的调度算法

Windows 2000/NT 中，基本的调度实体是线程。线程调度算法采用多级调度队列动态优先级算法。实现该调度算法的是内核中的一个数据结构，称为“调度程序数据库”。

每个线程都有一个基本执行优先级。该优先级介于进程基本优先级上下两个优先级的范围内。线程的动态优先级以线程的基本优先级为初值，随着进程所做工作的不同而

向上浮动。

Windows 2000 执行支持 32 个线程优先级，将这些优先级分成两类：实时优先级和可变优先级。实时线程的优先级为 16~31，是实时程序所用的高优先级线程。系统中大多数线程都是可变优先级线程，优先级为 1~15，其中，优先级 0 为系统所保留。

Windows 2000 是抢占式多任务的操作系统。如果一个较高优先级的线程由就绪状态变为要执行时，内核将中断或抢占其他线程的执行，被抢占的进程将被给予照顾，放在所在优先级队列的第一个。运行内核代码的线程不能被抢占，当内核运行时，它具有比系统中任何其他线程更高的优先级，从而保证内核有效地运行。尽管内核不能被抢先，但内核的大部分是可被高级中断所中断的。

8.2.6　Windows 2000 的内存管理

Windows 2000 的内存管理采用页式虚存。在功能和用户界面上，具有以下特点。

1. 提供 32 位虚地址

作为 32 位操作系统，Windows 2000 提供 32 位虚地址，因此，每个进程的虚地址空间最多可达 4GB（2^{32}B）。其中，低端 2GB 或 3GB 为用户地址空间，高端 2GB 或 1GB 为核心地址空间。其中的用户内存可被用户态和核心态线程存取访问，并且对每个进程都是不同的；而系统内存只能被核心态线程存取，并且对每个进程都是相同的。

2. 内存功能强大

支持大而稀疏的内存使用、内存映射文件、内存共享和写时复制。

3. 丰富强大的用户界面

通过丰富强大的用户界面，提供对环境子系统的支持，允许子系统以适当权限管理客户进程虚存以及对所有用户进程的支持，如允许进程分配、管理专用内存。环境子系统向它们的客户进程提供的内存进程视图，与一个本机进程的虚址空间并不总是对应的。

在内部实现上，内存管理主要由 Windows 2000 执行体中的虚存管理程序负责，并由环境子系统负责与具体 API 有关的一些用户态特性的实现。Windows 2000 的内存管理具有硬件相关代码分离和可移植性等先进特性。

8.2.7　Windows 2000 的文件系统

1. Windows 2000 文件系统的特点

Windows 2000 文件系统具有以下特点：

（1）兼容性和可扩充性。支持多个可装卸的不同类型的文件系统，如 Windows 个人系统、DOS 的 FAT 文件系统、OS/2 的高性能文件系统、CDROM 文件系统（CDFS）、NTFS 等。并支持文件系统类型之间的联机动态转换，如可将 HPFS 或 FAT 分区上的文件系统改为 NTFS 格式。

（2）功能强大。Windows 2000 文件系统提供了一个强大的本机文件系统：NTFS。

具有高的可靠性和安全性能，完全可恢复、磁盘镜像、磁盘复制、RAID 数据管理、带冗余校验的带区集，以及它的高安全特性和容错特性。对于现有的超大容量外存充分利

用，支持大文件和大文件系统。支持内存映像文件 I/O，支持硬链接和符号连接，文件压缩、多数据流文件、支持长文件名和多个扩展名（文件名及扩展名长度最大可为 255B）等。

2. Windows 2000 文件系统的实现

（1）Windows 2000 文件系统的整体实现机制采用了面向对象的模型，因此文件、目录等与其他资源一样，是作为对象来管理的。文件对象由 I/O 管理器进行管理。

（2）NTFS、FAT、HPFS 等文件系统，称为文件系统驱动程序，是作为 I/O 管理器中的一层驱动程序来实现的。

（3）用户打开文件表和系统打开文件表，在 Windows 2000 中体现为每个进程设置一个进程对象表以及它所指向的具体对象体。

8.2.8 Windows 2000 设备管理的特点

Windows 2000 中的设备管理具有以下特点：

（1）支持进程级异步 I/O 操作。

（2）具有与块设备管理相关的文件系统特点。映像文件 I/O，支持多个可装卸文件系统，借用虚存机制的新缓冲区实现方式等。

（3）允许系统管理员动态地向系统中增加或从系统中删除设备。

（4）提供服务程序，简化了设备驱动程序的开发工作，并能在多处理器系统中工作。允许用高级语言编写驱动程序。

（5）分层驱动程序模型提供可扩充性，使设备的动态增删更容易实现。

8.3 分布式操作系统

分布式系统泛指各种包含多个计算机（处理器）的信息处理系统，因此，并行计算机系统和网络系统也都属于分布式系统。

发展分布式系统的主要动力来自性能价格比方面的考虑。利用多台现成的机器协同工作，比建造一台更高性能的计算机成本更低一些。而且，大型的分布式系统的实际性能往往比一台单机要高得多。另外，许多应用本质上就是分布的，这是使用分布式系统的另一个原因。由于有多个 CPU，所以可把一个作业的多个任务分配到多个处理单元上进行处理，称为进程之间的并行性。

多处理机系统的当前趋势是研究并解决若干物理处理机的分布及处理问题。构造这样的系统有两种基本类型：第一种是紧密耦合系统，其中的处理机共享存储空间和时钟。在这类多处理机系统中，通信通常通过共享存储区进行；第二种是松散耦合系统，其中的处理机不共享存储区和时钟，每个处理机都有自己的局部存储器，这些处理机间的通信是通过各种通信线路，如高速总线或电话线等进行的。配置在分布式系统上的操作系统称为分布式操作系统。

8.3.1 分布式操作系统的特性

分布式操作系统具有以下特性。

1. 透明性

分布式系统中最重要的问题应该是如何实现单一的系统映像。也就是说，系统设计人员应该如何使用户相信自己所面对的不是一组用网络连起来的计算机，而是一部普通的分时电脑。系统的这一特性称为透明性。

透明性可以在两个层面实现。最简单的方式是在用户面前将分布性隐藏起来。在较低的层面上可以为程序提供透明性，如可以重新设计系统调用接口，使程序员看不到实际存在的多个处理机。

2. 可靠性

分布式系统比单机系统具有更高的可靠性。如果某台机器出故障了，可由别的机器继续完成它未完成的工作。整个系统的可用性在理论上是各台机器可靠性的逻辑或。

3. 高性能

分布式系统上运行程序的性能应比在单机上高。由于通信的存在，使得提高分布式系统的性能变得较为复杂。一般说来，包含大量彼此相关的小型计算的任务不太适合在通信速度较慢的分布式系统上运行，这样的问题常被称为细粒度的并行；而由一些相互作用较小的大计算量任务构成的作业称为粗粒度的并行。粗粒度的问题较适合在分布式系统中取得较高性能。

4. 伸缩性

应避免使用集中式的部件、表格和算法。分布式的算法特点为：没有一台机器具有关于整个系统的完整信息；机器只是根据本地信息作出决策；某台机器的故障不会导致整个算法的失败；不能隐含有关存在全局时钟的假设。

8.3.2　进程迁移

进程迁移就是将一个进程的状态从一台机器转移到另一台机器上，从而使该进程能在目标机上执行。这个概念主要来自于对大量互联系统负载平衡法的研究，使各个节点保持平衡的工作负载，加快计算速度，提高系统的性能。进程迁移主要有以下四方面的原因。

1. 负载共享

通过将进程从负载较重的节点迁移到负载较轻的节点，使系统负载达到平衡，从而提高整个系统的效率。

2. 通信性能

可以将相互间紧密作用的进程迁移到同一节点，以降低它们相互作用期间的通信开销。类似地，当某进程在执行数据分析时，如果它所访问的文件远远大于进程本身，则此时应将该进程迁移到文件所驻留的系统中去。

3. 可获得性

运行时间较长的进程在出现错误时可能需要迁移。一个想继续的进程既可以迁移到另外的系统，也可以推迟运行，待错误恢复后可在当前系统中重新开始。这些情况都需要分布式操作系统的调度和处理。

4. 利用特定资源

一个进程可以迁移到某特定的节点上，以利用该节点上独有的硬件或软件资源。

为了实现进程迁移，在分布式系统中建立了相应的进程迁移机制。该机制主要用于解决如下问题：由谁来发动进程迁移、迁移进程的哪些部分、迁移的具体实施方案、对未完成的报文和消息的处理方式。

8.3.3 分布式进程管理

在集中式方式系统中，所有的进程都存在于同一个系统中，它们共享所有的集中式资源，如内存等，因而也可以共享信号量、时钟、锁等。然而，在分布式操作系统中，各处理机相互隔离，没有共享内存，因此，在集中式方式系统中所采用的进程同步方式已不再适用。在分布式系统中，要实现进程的同步比在集中式系统中的进程同步复杂得多。实现进程同步，主要采用了以下几种方法。

1. 事件定序

这种算法将事发前关系扩充成为系统中所有事件的全序关系。

事发前关系：因为进程内部执行的程序是有序的，并且在各自的处理机上运行，因此在进程的执行中所有的事件都是有序的。根据因果关系原理，一条消息仅当已发送以后才可被接收。

在一个事件集合上定义事发前关系如下：

若 A 和 B 是同一进程中的两个事件，且 A 在 B 之前执行，则可表示为：A→B。

若 A→B 且 B→C，则 A→C。

这里，每个事件给一个时间戳。对每一个事件 A、B，若 A→B，则 A 的时间戳<B 的时间戳。若为每个进程定义一个逻辑时钟 LC_i，那么有 $LC_i(A)<LC_i(B)$。一个事件的时间戳就是有关那个事件的逻辑时钟的值。

2. Lamport 算法

该算法基于分布式队列的概念，并含有以下假定：①分布式系统由 N 个站点组成，每个站点都有惟一的编号 1～N。每个站点都只有一个进程负责进程互斥访问资源。②按发送的顺序接收从一个进程到另一个进程的消息。③每条消息都在有限时间内正确地送到目的地。④网络是全互连的，任意两个进程间可以直接传递消息，不需另一个节点中转。

Lamport 算法如下：

（1）当进程 P_i 请求访问某个资源时，该进程把请求消息挂在自己的请求队列上，并发送一个 Request(T_i,i)消息给所有其他进程。

（2）当进程 P_j 收到 Request(T_i,i)消息时，将这个消息放入自身数组 q[i]中，并传送 reply(T_j,j)给所有其他进程。

（3）进程 P_i 满足以下两个条件时就可以访问一个资源。

条件一：P_j 自身请求访问该资源的消息已处于请求队列的最前面。

条件二：P_j 已接收到从所有其他进程发来的响应消息，这些响应消息上邮戳的时间晚于(T_i,i)。

这就说明，所有其他进程或者都不访问该资源，或者要求访问，但其时间较晚。

（4）P_i 通过发送 Release(T_i,i)消息来释放它所占用的资源。该消息也置入其自身的数组项中，并传递给所有其他进程。

（5）当进程 P_j 收到进程 P_i 的 Release 消息后，便从自己的队列中消去 P_i 的 Release(T_i,i)消息。

为保证互斥，该算法需要 3(N-1)条消息，其中(N-1)个 Request 消息，(N-1)个 Reply 消息以及(N-1)个 Release 消息。

这种算法满足互斥要求，且公平、无死锁，不会产生饥饿。

3. Ricart 算法

Ricart 算法对 Lamport 算法进行了一些简化，希望将 Release 消息删去以优化原始算法。Ricart 算法与 Lamport 算法的假设相同。

每个站点都有一个进程负责控制资源的分配。该进程有一个数组 q 并遵循以下规则：

（1）当进程 P_i 请求访问资源时，它会发出一个请求 Request(T_i,i)。时戳为当前本地时钟的值。将这条消息放入自身数组 q[i]中，然后将消息发送给所有其他进程。

（2）当进程 P_j 收到 Request(T_i,i)后，按下列规则进行处理。

1）如果 P_j 正处于临界段，则延迟发送 Reply 消息。

2）如果 P_j 并不等待进入临界段，就发送 Reply t(T_j,j)消息给所有其他进程。

3）如果 P_j 等待进入其临界段，且收到的消息在 P_j 的 Request 之后，则将收到的消息放入其数组的 q[i]中，并延迟发送 Reply 消息。

4）如果 P_j 等待进入其临界段，但收到的消息在 P_j 的 Request 之前，则将收到的消息放入其数组的 q[i]中，并发送 Reply 消息给进程 Pi。

（3）如果进程 P_i 从所有其他进程都收到了 Reply 消息，它就可以访问资源进入该临界段。

（4）当进程 P_i 离开临界段时，它会给每个挂起的 Request 发送一个 Reply 消息，从而释放资源。

在本算法中，需要 2(N-1)条消息，其中(N-1)个 Request 消息，表示进程 P_i 要进入临界段；(N-1)个 Reply 消息以允许其他进程的访问。

本算法利用时间戳来实现进程的同步，可以避免死锁及饥饿。

4. 令牌方法

令牌本身是一种特定格式的报文，通常长度为 1B。它是在系统设置的，用以实现进程的互斥及象征存取权利。它不断地在由进程组成的逻辑环中循环，环中的每一个进程都有惟一的前趋和后继。

利用令牌实现互斥。当环中的令牌循环到某进程并被接收，如果该进程希望进入其临界区时，它便保持该令牌，进入临界区。退出临界区时，又把令牌传送给它的后继进程。如果收到令牌的进程并不要求进入临界区，便直接将令牌再传送给它的后继。由于逻辑环中只有一个令牌，所以每次也只能有一个进程进入临界区，从而实现了进程的互斥。

在使用令牌传送时，必须满足以下两点：

（1）逻辑环应能够及时发现环路中某进程失效或退出，以及通信链路的故障。一旦发现这种进程或故障，应立即撤消该进程，并对逻辑环进行重构。

（2）保证逻辑环中在任何时候都有令牌在循环。一旦发现令牌丢失，应立即选择一个进程，用来产生一个新的令牌。

本章小结

本章主要介绍了目前常用的操作系统 UNIX、Windows 系统的内核结构，对分布式操作系统作了简要介绍，同时对于操作系统的安全性提出了要求。

UNIX 系统具有多用户多任务、可移植性、树形文件系统结构、I/O 重定向和管道技术、丰富的实用程序、电子邮件等特点，属于层次结构的操作系统模型。

UNIX 系统中，进程的结构也称为进程实体，由三部分构成：用户级上下文、寄存器上下文和系统级上下文。该系统是分时系统，它的进程调度采用动态优先数轮转调度算法。进程的状态共有 9 个，它们在一定的情况下可以转换。内存管理采用求请调页存储管理方式，支持内外存的对换功能。UNIX 系统将文件分为三类，即普通文件、目录文件和特别文件（即设备文件）。UNIX 系统的目录结构为有向非循环图结构。UNIX 系统中，文件是以块为单位存放在介质上的。文件目录由文件名和该文件的索引节点号构成。其中，文件名占 14 个字节，索引节点号（或索引节点指针）占两个字节。文件的物理结构采用混合索引方式，对于长度较小的文件具有较快的读写速度，同时又具有支持大文件的功能。对于空闲盘块的管理，采用成组链接法。系统为打开文件建立的数据结构有三个，即用户文件描述符表、文件表和内存索引节点。把设备分为两类，即字符设备和块设备。

Windows 2000 操作系统具有多进程、多线程的特点，采用了客户/服务器模型。进程由以下几部分组成：程序、该进程专用的地址空间、系统资源、该进程的线程。线程由以下几部分构成：线程标识符、一组易变寄存器、两个栈和专用的存储区域。

Windows 2000/NT 中，基本的调度实体是线程。线程调度算法采用多级调度队列动态优先级算法。内存管理采用页式虚存。文件系统的整体实现机制采用了面向对象的模型。

分布式系统泛指各种包含多个计算机（处理器）的信息处理系统。多处理机系统有两种基本类型：第一种是紧密耦合系统，第二种是松散耦合系统。配置在分布式系统上的操作系统称为分布式操作系统。

分布式操作系统相对于集中式操作系统具有透明性、可靠性、高性能、伸缩性的特点。进程迁移是分布式系统区别于其他系统的一个非常重要的功能。分布式进程管理及实现进程同步、互斥主要采用了以下方法：事件定序、Lamport 算法、Ricart 算法、令牌方法。

习 题 八

8-1 说明 UNIX 系统具有哪些特点。

8-2 说明 UNIX 系统的进程调度算法。

8-3　说明 UNIX 系统的进程状态及其转换。

8-4　说明 UNIX 系统的内存管理方式。

8-5　说明 UNIX 系统与 Linux 系统在内核管理方面的异同点。

8-6　说明 UNIX 系统的文件系统结构及按名查找方式。

8-7　Windows 操作系统具有哪些特点？

8-8　说明 Windows 操作系统的内核管理方式。

8-9　什么是分布式操作系统？包括哪两种不同的类型？

8-10　分布式系统区别于网络系统的特点是什么？

8-11　分布式系统如何实现进程同步？有哪几种算法？

8-12　操作系统的安全性包含哪些方面？

8-13　用哪些方面实现操作系统的安全保护机制？

8-14　病毒防御机制可采取哪些方法实现？

第 9 章　操作系统的安全性

本章学习目标

随着计算机应用的日益广泛，人们对计算机系统的安全性也越来越重视。在维护计算机系统安全的众多措施中，操作系统的安全性占有重要地位。本章初步讲述了操作系统安全性方面的一些问题，通过本章的学习，读者应掌握以下内容：

- 计算机系统安全性的内涵
- 操作系统的安全性功能
- 操作系统的安全机制
- 安全操作系统的开发

9.1　操作系统安全性概述

随着计算机应用的日益广泛和深入，人们对计算机的依赖越来越大。通常，政府机关和企事业单位，都将大量的重要信息存储在计算机系统中；随着计算机网络应用的普及，人们在日常的工作、生活中越来越多地通过计算机网络来交流信息、处理事务，以及从事其他各种活动，如商品交易、游戏、电子支付等。人们在享用计算机技术带来的工作和生活便利的同时，要求计算机系统安全可靠，这是人们使用计算机的一个前提条件。不安全又谈何便利，不但没有便利可言，而且会极大地损害人们的利益。人们要求存储在计算机系统中的大量重要信息是安全的、信息在计算机网络上的传输是安全的。在计算机系统中，操作系统是控制中心，所以操作系统的安全性是其他软件安全职能的根基，缺乏这个安全的根基，构筑在其上的应用系统以及安全系统的安全性就得不到根本保障。一个有效可靠的操作系统应具有良好的安全性能，可提供必要的安全保障措施。

9.1.1　计算机系统安全性的内涵

1.　对计算机系统安全性的威胁

（1）自然灾害。如停电、火灾、洪水、地震、战争或者计算机系统所处的其他恶劣环境等都会对计算机造成一定的损害。计算机本身不能承受强列的震荡及强力冲击，另外，设备对环境如温度、湿度等的要求也较高。

（2）计算机系统自身的软硬件故障。如软盘或硬盘的损坏导致数据无法读出、网络通信错误、程序设计中的缺陷等。

（3）合法用户的使用不当。如数据输入错误、软硬件安装、设置错误、丢失保存数

据的介质等。

（4）非法用户对计算机系统的攻击。非法用户往往利用计算机系统的弱点来达到他们自己的目的。如各种行业的间谍可能对系统的安全性造成威胁；有一些恶作剧者可能编制一些病毒等，也会对系统的安全造成一定的威胁。

2. 计算机系统安全性的内涵

不同的系统对操作系统的安全性有不同的要求，但一般来说，一个安全的计算机系统应具有下面三个特性：

（1）保密性。指系统不受外界破坏、无泄露、对各种非法进入和信息窃取具有防范能力。只有授权用户才能存取系统的资源和信息。

（2）完整性。指信息必须按照其原型保存，不能被有意或无意地修改，只有授权用户才能修改（对软件或数据未经授权的修改都可能导致系统的致命错误）。完整性可分为软件完整性和数据完整性。

（3）可用性。可用性是指对合法用户而言，无论何时，只要需要，信息必须是可用的，授权用户的合法请求能准确及时地得到服务或响应，不能对合法授权用户的存取权限进行额外的限制。

9.1.2 操作系统的安全性

因为操作系统是计算机系统的核心软件，由操作系统对计算机系统的所有资源进行统一管理，所以操作系统的安全性是保障计算机系统安全性的一个重要因素。非法用户攻破操作系统的防范就可以得到计算机系统保密信息的存取权。因此，操作系统从安全性方面考虑，应具有以下功能：

（1）有选择的访问控制。有选择的访问控制包括使用多种不同的方式来限制计算机环境下对特定对象的访问，对计算机级的访问可以通过用户名和密码组合及物理限制来控制，对目录或文件级的访问则可以由用户和组策略来控制。在操作系统设计的初期定义有选择的访问控制是很重要的。

（2）内存管理与对象重用。内存管理是操作系统安全中的一个重要组成部分。在复杂的虚拟内存管理器出现之前，将含有机密信息的内容保存在内存中风险很大。系统中的内存管理器必须能够隔离开每个不同进程所使用的内存。在进程终止且内存被重用之前必须在再次访问它之前，将其中的内容清空。

（3）审计能力。安全系统应该具备审计能力，以便测试其完整性，并可追踪任何可能的安全破坏活动。审计功能至少包括可配置的事件跟踪能力、事件浏览和报表功能、审计事件、审计日志访问等。

（4）加密的数据传送。数据传送加密保证了在网络传送时所截获的信息不能被未经身份认证的代理所访问。针对窃听和篡改，加密密钥具有很强的保护作用。

（5）加密的文件系统。对文件系统加密保证了数据只能被具有正确选择访问权的用户所访问。数据加密和解密的方式对用户来说应是透明的。

（6）安全的进程间通信机制。进程间通信也是给系统安全带来威胁的一个主要因素，

应对进程间的通信机制做一些必要的安全检查，禁止高安全等级进程通过进程间通信的方式传递信息给低安全等级进程。

9.1.3 计算机系统安全性评价标准

为了能有效地以工业化方式构造可信任的安全产品，必须建立对该产品进行安全性评价的标准。

1. TCSEC 标准

美国国防部在 20 世纪 80 年代中期制定了一组计算机系统安全需求标准，共包括 20 多个文件，每个文件分别使用不同颜色的封面，统称为“彩虹系列”。其中最核心的是具有橙色封面的“可信任计算机系统评价标准（TCSEC—Trusted Computer System Evaluation Criteria）”，称为“橙皮书”。

TCSEC 将计算机系统的安全程度分成 D、C、B、A 四个等级，每个等级又包含一个或多个级别。共包括 8 个安全级别：D、C1、C2、B1、B2、B3、A1、A1，这 8 个级别渐次增强。

（1）D 等。D 等只有一个级别：D1 级。D1 级是计算机安全的最低级，整个计算机系统是不可信任的，硬件和操作系统都很容易被侵袭。另外，D1 级计算机系统标准规定不对用户进行验证，即任何人都可以自由地使用该计算机系统。达到 D1 级的操作系统有 DOS、Windows 3.x、Windows 95（不在工作组方式中）、Apple 的 System 7.x 等。

（2）C 等。C 等为自主型保护，由 C1 和 C2 两个级别组成。

C1 级是无条件安全防护系统，要求硬件有一定的安全保护，如硬件有带锁装置，需要钥匙才能使用计算机。用户在使用计算机系统前必须先登录。另外，作为 C1 级保护的一部分，无条件访问控制允许系统管理员为一些程序和数据设立访问许可权限。常见的 C1 级操作系统有 UNIX、XENIX、Novell 3.x 或更高版本、Windows NT。

C1 级防护的不足之处在于用户直接访问操作系统的根。C1 级不能控制进入系统的用户的访问级别，所以用户可将系统中的数据任意移走，还可以更改系统的配置，获取比系统管理员允许的更高权限。

C2 级对上述 C1 级的不足之处做了补充，引进了受控访问环境（用户权限级别）的增强特性。这一特性以用户权限为基础，进一步限制了用户执行某些系统指令。用户权限以个人为单位授权用户对某一目录进行访问，如果其他程序和数据在同一目录下，那么用户也将自动获得访问这些信息的权限。

授权分级使系统管理员能够给用户分组，授予他们访问某些程序或访问分级目录的权限。

C2 级系统还采用了系统审计。审计特性跟踪所有的“安全事件”，如登录（成功的和失败的）以及系统管理员的工作（如改变用户访问权限和密码）等。达到 C2 级的常见操作系统有 UNIX、XENIX、Novell 3.x 或更高版本、Windows NT。

（3）B 等。B 类为强制型保护，由三个级别组成。

B1 级称为“标记安全防护”级，支持多级安全，它满足 C2 级所有的要求。“标记”

指网上的一个对象，该对象在安全防护计划中是可识别且受保护的。“多级”是指这一安全防护装在不同级别（如网络、应用程序和工作站等），对敏感信息提供更高级的保护。

安全级别分为保密和绝密，在计算机中有“特务”成员，如国防部和国家安全局系统。在这一级中，对象（如磁盘、文件目录等）必须在访问控制之下，不允许拥有者修改他们的权限。

B1 级安全措施的计算机系统随操作系统而定。政府机构和防御承包商是 B1 级计算机系统的主要拥有者。目前国内达到 B1 级的操作系统有红旗安全操作系统 2.0 版、南京大学的 SoftOS 等。

B2 级称为“结构化防护”，要求计算机系统中所有对象加标签，而且给设备（如工作站、终端和磁盘驱动器）分配安全级别。例如，可以允许用户访问一台工作站，但不允许访问含有职员工资资料的磁盘子系统。

B3 级称为“安全域”。要求用户工作站或终端通过可信任途径连接网络系统，而且这一级采用硬件保护安全系统的存储区。这一级支持安全管理者的实现，审计机制能实时报告系统的安全性事件，支持系统恢复。

（4）A 等。A 类为“验证型保护”，由两个级别组成。

A1 级：从实现的功能上看，它等同于 B3 级，它的特色在于形式化的顶层设计规格、形式化验证与形式化模型的一致性和由此带来的更高的可信度。并且这一级还附加一个安全系统受监视的设计要求，合格的安全个体必须分析并通过这个设计要求。A1 级要求构成系统的所有部件来源必须有安全保证，以此保障系统的完善与安全。例如，在 A 级设置中，一个磁盘驱动器从生产厂房直至销售到计算机房的过程中都被严格跟踪。

A1 级以上：比 A1 级可信度更高的系统归入该级。

2. CC 标准

1991 年，在欧洲共同体的赞助下，英、德、法、荷四国制定了拟为欧共体成员国使用的共同标准——信息技术安全评定标准（ITSEC）。随着各种标准的推出和安全技术产品的发展，迫切需要制定一个统一的国际标准。美国和加拿大及欧共体国家一起制定了一个共同的标准，于 1999 年 7 月通过了国际标准组织的认可，确立为国际标准，简称为 CC——Information Technology Security Evaluation Common Criteria。

CC 本身由两个部分组成，一部分是一组信息技术产品的安全功能需要定义，另一部分是对安全保证需求的定义。

安全功能需求部分是按结构化方式组织起来的安全功能定义，分为类（Class）、族（Family）和组件（Component）三层。每个类侧重一个安全主题，CC 共包括 11 个类，基本上覆盖了目前安全功能的所有方面。每个类包含了一个或多个族，每个族基于相同的安全目标，但侧重方面和保护强度有所不同。每个族包含了一个或多个组件。一个组件确定了一组最小可选择的安全需求集合，即在从 CC 中选择安全功能时，不能对组件再做拆分。一个族中的组件排列顺序代表强度和能力的不同级别。

安全保证需求组织方式与安全功能需求相同，即按“类—族—组件”方式结构化地定义了各种安全保证的需求，共包括 10 个类。为了能够有效地使用安全功能需求和安全保

证需求，CC 还引入了“包（Package）”的概念，以提高已定义结果的可重用性。在安全保证需求之中，特别以包的概念定义了 7 个安全保证级别（EAL）。这 7 个级别分别定义如下。

EAL1：功能性测试级，证明 TOE 与功能规格的一致。

EAL2：结构性测试级，证明 TOE 与系统层次设计概念的一致。

EAL3：工程方法上的测试及校验级，证明 TOE 在设计上采用了积极安全的工程方法。

EAL4：工程方法上的方法设计、测试和评审级，证明 TOE 采用了基于良好的开发过程的安全工程方法。

EAL5：半形式化设计和测试级，证明 TOE 采用了基于严格的过程的安全工程方法并适度应用了专家安全工程技术。

EAL6：半形式化地验证设计和测试级，证明 TOE 通过将安全工程技术应用到严格的开发环境中来达到消除大风险保护高价值资产。

EAL7：形式化地验证设计和测试级，证明 TOE 所有安全功能经得起全面的形式化分析。

安全保证级别测试并未对产品增加任何安全性，仅仅是告诉用户产品在多大程度上是可信的。一般而言，安全要求越高、威胁越大的环境，应采用更可信的产品。

3. 《计算机信息系统安全保护等级划分准则》

中国于 1999 年颁布的《计算机信息系统安全保护等级划分准则》（GB 17859-1999）将计算机信息系统安全程度划分为以下五个等级：

第 1 级为用户自主保护级。本级的安全保护机制使用户具备自主安全保护的能力，保护用户和用户组信息，避免其他用户对数据的非法读写和破坏。

第 2 级为系统审计保护级。本级的安全保护机制具备第一级的所有安全保护功能，并创建、维护访问审计跟踪记录，以记录与系统安全相关事件发生的日期、时间、用户和事件类型等信息，使所有用户对自己行为的合法性负责。

第 3 级为安全标记保护级。本级的安全保护机制具有系统审计保护级的所有功能，并为访问者和访问对象指定安全标记，以访问对象标记的安全级别限制访问者的访问权限，实现对访问对象的强制保护。目前国内达到第 3 级的安全操作系统有：红旗安全操作系统 2.0 版、南京大学的 SoftOS 等。

第 4 级为结构化保护级。本级具备第 3 级的所有安全功能，并将安全保护机制划分成关键部分和非关键部分相结合的结构，其中关键部分直接控制访问者对访问对象的存取。本级具有相当强的抗渗透能力。

第 5 级为安全域级保护级。本级的安全保护机制具备第 4 级的所有功能，并特别增设访问验证功能，负责仲裁访问者对访问对象的所有访问活动。本级具有极强的抗渗透能力。

9.2 操作系统的安全机制

操作系统的安全机制的功能是防止非法用户登录计算机系统，防止合法用户非法使用计算机系统资源以及加密在网络上传输的信息，防止外来的恶意攻击。总之是防止对计算

机系统本地资源和网络资源的非法访问。

9.2.1　内存保护机制

内存的保护相对是一个比较特殊的问题。在多道程序中，一个重要的问题是防止一道程序在存储和运行时影响到其他程序。操作系统可以在硬件中有效地使用硬保护机制进行存储器的安全保护，现在比较常用的有界址、界限寄存器、重定位、特征位、分段、分页和段页式机制等。

为将进程的内存空间分开，许多系统采用虚拟内存策略来实现。分段、分页或两者相结合，可提供一个管理内存的有效方法。如果进程完全分开，那么操作系统必须确保每段或每页只被其所属进程存取，这可以通过在页表或段表中无重复项来实现。

如果允许共享，那么同一段或页可在不止一个表中出现。这种共享在分段或段页结合的系统中最容易实现。在这种情况下，段结构对应用程序可见，应用程序可将段说明成共享或非共享。在分页环境中，由于内存结构对用户透明，因此区别共享和非共享内存很困难。

9.2.2　用户身份认证机制

操作系统的许多保护措施大都基于鉴别系统的合法用户，身份鉴别是操作系统中相当重要的一个方面，也是用户获取权限的关键。为防止非法用户存取系统资源，许多操作系统采取了切实可行的、极为严密的安全措施。目前最常用的用户身份认证机制是口令。此外，数字签名、指纹识别、声音识别等操作系统安全机制也逐渐在投入使用。

1. 口令

口令是计算机系统和用户双方都知道的某个关键字，相当于是一个约定的编码单词或“暗号”。它一般由字母、数字和其他符号组成，在不同的系统中，其长度和格式也可能不同（如大小写是否敏感等）。口令的产生既可以由系统自动产生，也可以由用户自己选择。

口令在使用时一般和另一个标识——用户 ID 一起使用。用户 ID 是可以公开的，但口令是保密的，否则就失去了口令的意义。当系统要求输入用户 ID 和口令时，就可以根据要求在适当的位置进行输入，输入完毕确认后，系统就会与口令文件中的口令进行比较匹配，若一致，则通过验证；否则拒绝登录或再次提供机会让用户进行登录。

2. 口令使用的安全性

由于口令的位数是有限的，而组成口令的字符也是有限的，所以在理论上，任何的口令都可以破解。因此，口令作为保护是有限的。另外，许多非法入侵者会采用各种手段窃取用户口令，如攻击口令文件，或者用特洛伊木马伪装成登录界面骗取用户的口令等。

所以，用户在设置和使用口令时要注意以下问题：

（1）口令要尽可能长，这样要猜出口令就需要很长时间，其可能性就小。操作系统在这方面也有要求，如要求口令的长度至少为 8 位等。

（2）多用混合型的口令，即其中同时有字母、数字和其他字符。

（3）不要用自己或家人的生日、姓名、常用单词等作为口令。许多非法入侵者猜测口令时会首先使用这些具有强烈特征的字符串作为口令来尝试。

（4）经常更换口令。许多操作系统也有要求，在规定的时间内更改口令，否则口令失效。最极端的方法是使用一次性口令。这时，用户有一本口令书，记着一长串口令，登录时每次都采用书中的下一个口令。如果入侵者破译出口令，也没有什么用，因为用户下一次就会用另一个口令。这种做法的前提是用户必须谨防口令书的丢失。

（5）设置错误口令注册次数（如许多操作系统允许的错误次数为 3 次），一旦超过这个次数就无法注册登录，只有系统管理员才能使之恢复正常。

（6）用户在使用系统前，要确认系统的合法性，以免被骗取口令。现在的操作系统都提供了一些手段以确保用户是在真实系统中进行登录，如 Windows NT 中按 Ctrl+Alt+Del 组合键才开始登录。

3. 系统口令表的安全性

（1）限制明文系统口令表的存取。为了验证口令，系统必须采用将用户输入的口令和保存在系统中的口令相比较的方式，攻击者可能攻击的目标是口令文件，借助于系统口令表可以正确无误地获取口令。

在某些系统中（如 UNIX），口令表是一个文件，实际上是一个由用户标识及相应口令组成的列表。显然，不能让任何人都能访问到该表，为此，系统采用了不同的安全方法来保证。保护口令表的安全机制是使用强制存取控制，限制它仅可为操作系统所存取。更进一步的是，只允许那些需要存取该表的操作系统模块存取。

（2）加密口令文件。加密口令文件表较为安全，这样，读文件内容对入侵者来说，还必须经过解密才有用，增加了破解的难度。一般使用传统加密及单向加密这两种加密口令的方法。

使用传统加密方法，整个口令表被加密，或只加密口令部分。当接收到用户输入的口令时，所存取的口令被解密。使用这种方法在某一瞬间会在内存中得到用户口令的明文，有可能被人窃取，显然这是一个缺陷。

另一个较安全的方法是使用单向加密。加密方法相对简单，解密则是用加密函数。在用户输入口令时，口令被加密，然后将两种加密形式进行比较，若相同，则成功通过验证。

以一种伪装的形式保存口令表可以进一步提高安全性，当然存取的方式仍然限制为具有合法需要的进程。

4. 物理鉴定

检查用户是否有某些特定的“证件”是另一种不同的认证方法，一般用磁卡或 IC 卡。卡片插入终端，系统可以查出卡片所有者，卡片一般和口令一起配合工作，用户要登录成功，必须有卡片，并且知道密码。银行的 ATM 就是这样工作的。

测量那些难以伪造的特征也是一种方法。如终端上的指纹或声波波纹读取机可验证用户身份，还可直接用视觉辨认，当然这种认证方法对于终端设备的要求比较高。

签名分析是另一种技术。用户采用与终端相连的特殊笔签名后，计算机与在线已知样本进行比较。更好的方法是不比较签名，而是比较笔签名时笔的移动情况，模仿者或许可以模仿签名，但在签名时确切的行笔顺序，他就不了解了。

这些物理鉴别方法涉及的另一个问题是，用户可能不能接受。目前一般是在比较重要、

保密要求较高的系统中会采用物理鉴别手段。

9.2.3　访问控制技术

1. 保护域

为有助于理解各种各样的保护方法，我们引入了“保护域”的概念。每一个进程在任何时刻运行在一个保护域内，该保护域具体指定了该进程可以存取的资源。即每一个域定义了一系列的对象以及在对象上可执行的操作（即存取权限），所以域是<对象，权限>对的集合，每个对标记了一个对象和一个可执行操作的子集。

例如，域 D 有存取权限<文件 F，{读，写}>，那么，在域 D 中运行的进程对文件 F 既可以读，也可以写；如果不在域 D 中的进程，那只能对文件 F 执行其他的操作。

多个域可以是有交集的，它们可以共享相同的存取权限。如图 9-1 所示，有三个域 D1、D2、D3，存取权限<f4,{print}>由域 D2 和 D3 共享，这意味着一个进程只要运行在域 D2 或 D3 中，都可以打印对象 f4。虽然在图中没有标出，但一个对象可以在多个域中出现，并且权限在不同域中也可以不同（图中，R 代表读，W 代表写，X 代表执行，P 代表打印，下同）。

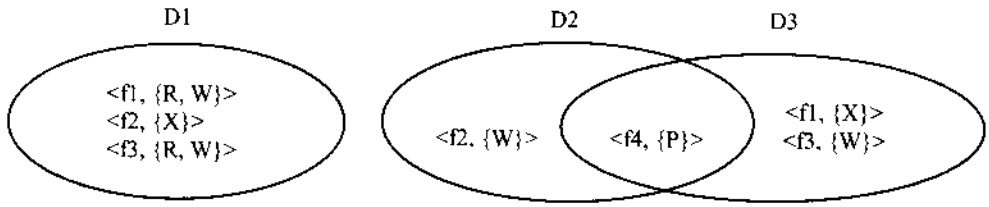


图 9-1　3 个保护域

进程和域之间的关系可以是静态的，也可以是动态的。如果是静态的，则必须施加“须知原则”，并建立一个机制来改变域的内容。例如，一个进程在运行时可能有两个不同的阶段，在第一个阶段需要“读”权限，在另一个阶段则需要“写”权限。如果域是静态的，则应定义该域同时具有写和读权限。然而，这种做法使得进程的每一个运行阶段都拥有比必须的权限较多的权限，即当仅需要读权限时，还拥有写权限，反过来也是这样。这样就违反了“须知原则”，所以必须允许域的内容是可变的，这样才能体现“最小权限原则”。

如果是动态关系，同样也有一个机制允许进程从一个域切换到另一个域，也允许域的内容进行修改。如果不能修改域的内容，则可以通过创建一个具有新内容的域，然后进程再切换到该新域。

域可以有以下几种实现方式：

（1）每一个用户可以是一个域。在这种情形下，可以存取的对象及其权限取决于用户的身份，当一个用户注销而另一用户登录时，就进行域的切换。

（2）每一个进程可以是一个域。此时，可以存取的对象及其权限取决于进程的标志，当一个进程向另一个进程发送消息并等待回应时，就进行域的切换。

（3）每一个过程也可以是一个域。此时，可以存取的对象及其权限取决于过程内定

义的局部变量，当进行过程调用时，就进行域的切换。

2. 存取矩阵

有了域的概念后，系统如何来记录某个对象属于哪个域呢？从抽象的角度，这里介绍矩阵方法。其中，矩阵的行表示域，列表示对象，矩阵内的每一项表示存取权限的集合。如表 9-1 所示，给定该矩阵及当前域号，系统可以给出是否能从该指定域中按特定方式访问某对象。

表 9-1 一个存取矩阵

对象 / 域	文件 1	文件 2	文件 3	打印机
D1	R，W	X	R，W	
D2		W		P
D3	X		W	P

如果注意到域本身也是对象，就可以把域切换包含在矩阵模型中。表 9-2 重新给出了表 9-1 的内容，只增加了作为对象的三个域，并增加了“切换”操作（Switch）。表中域 D1 可以切换到域 D2，但不能返回。

表 9-2 存取矩阵把域作为对象

对象 / 域	文件 1	文件 2	文件 3	打印机	域 D1	域 D2	域 D3
D1	R，W	X	R，W			Switch	
D2		W		P			
D3	X		W	P	Switch		

3. 存取矩阵的实现

在实际应用中，实际上很少存储如表 9-2 所示的矩阵，该矩阵极大，并且有多个空项，大多数域都只存取很少的对象。所以存储这样一个大而空的矩阵，将导致大量的磁盘空间浪费。下面介绍两种比较实用的方法：按行或按列来存储矩阵，且只存储非空元素。

（1）存取控制表。将存取矩阵按列存放，这样每个对象被赋予一张排序的列表，其中列出了可以访问该对象的全部域以及怎样访问，这张表就称为存取控制表（Access Control List），简称为 ACL。具体实现时，可把每个文件的 ACL 放在磁盘的一个单独块中，并在文件的目录区链接这个磁盘块的块号。因为只存储了非空项，所需的全部 ACL 存储空间比整个矩阵存储所需的空间要少得多。例如，对应于表 9-1 的存取控制表可以这样描述：

文件 1：(D1,RW), (D3,X)

文件 2：(D1,X), (D2,W)

文件 3：(D1,RW), (D3,W)

打印机：(D2,P), (D3,P)

对象的所有者能随时改变对象的 ACL，可以方便禁止原先允许的访问。可能的问题是，

ACL 改变很有可能不影响当前正在使用该对象的用户。

（2）访问权限表。将存取矩阵按行存放，这样每个域都赋予一张在该域内可能访问的对象表以及每个对象允许进行的操作，该表就称为访问权限表，其中的每一项叫作权限。

一张访问权限表的描述如表 9-3 所示，其中描述了表 9-1 中域 D1 的权限。表中给出了对象的类型以及对这个对象允许执行的合法操作，并给出指向对象本身的指针。访问权限表本身也是对象，可以从别的访问权限表中引用，因此很容易实现了域的共享。

表 9-3　访问权限表

编号	对象类型	权限	对象指针
1	文件	RW-	指向文件 1
2	文件	--X	指向文件 2
3	文件	RW-	指向文件 3

除了依赖于具体对象的权限，如读、写和执行，往往还有用于全部对象的一般权限，如复制权限、复制对象、删除权限、删除对象等。

另外，在使用访问权限表的系统中，很难撤消对某对象的访问权，系统很难找出全部权限并收回，因为这些权限事实上存储在遍及磁盘的访问权限表中。

9.2.4　加密技术

加密技术将信息编码成如密码文本一样含义模糊的形式。在现代计算机系统中，加密技术越来越重要。在网络化的计算机系统中，要想提供一种机制使信息不可访问很困难，所以，将信息加密成另一种形式，如果没有解密，即使访问到它，其内容也是不可识别的。加密技术的关键是能够有效地生成密码，使它基本上不可能被未授权的用户解密。

数据加密的模型基本上由以下四部分构成。

（1）明文。需要被加密的文本，称为明文 P。

（2）密文。加密后的文本，称为密文 Y。

（3）加密、解密算法 E、D。用于实现从明文到密文，或从密文到明文的转换公式、规则或程序。

（4）密钥 K。密钥是加密和解密算法中的关键参数。

加密过程可描述为：明文 P 在发送方经加密算法 E 变成密文 Y。接收方通过密钥 K，将密文转换为明文 P。

加密具体有很多种实现方法，如简单的易位法、置换法、对称加密算法和非对称加密算法等。

9.2.5　病毒及其防御机制

1. 计算机病毒概述

计算机病毒是一种可传染其他程序的程序，它通过修改其他进程使之成为含有病毒的版本或可能的演化版本。病毒可经过计算机系统或计算机网络进行传播。一旦病毒进入了

某个程序，将会影响该程序的运行，并且这个受感染的程序可以作为传染源，继续感染其他程序，甚至对系统的安全性造成威胁。

计算机病毒大致由三部分构成。

（1）引导模块。负责将病毒引导到内存，对相应的存储空间实施保护，以防止其他程序覆盖，并且修改一些必要的参数，为激活病毒做准备工作。

（2）传染模块。主要负责将病毒传染给其他计算机程序，它是整个病毒程序的核心，由两部分构成：一部分判断是否具备传染条件，一部分具体实施传染。

（3）发作模块。主要包括两部分，一部分负责病毒触发条件的判断，另一部分负责病毒危害的实施。

2. 病毒防御机制

病毒对计算机系统的传播及危害，对计算机系统的安全造成极大的威胁，它的传染性主要与操作系统有关。

病毒可在不同介质之间传播，一个比较普遍的例子是通过磁盘传播。病毒嵌入某个合法程序隐藏起来，当这个程序被存入磁盘时，病毒也就随之被复制到了磁盘。由于这个磁盘存储的过程一般是通过操作系统的磁盘操作功能来实现的，不同操作系统的磁盘操作功能并不相同。所以一般情况下，针对某种操作系统的病毒不能感染其他互不兼容的操作系统。

病毒防御措施通常将系统的存取控制、实体保护等安全机制结合起来，通过专门的防御程序模块为计算机建立病毒的免疫系统和报警系统。防御的重点在操作系统敏感的数据结构、文件系统数据存储结构和 I/O 设备驱动结构上。这些敏感的数据结构包括：系统进程表、关键缓冲区、共享数据段、系统记录、中断向量表和指针表等。很多病毒试图修改甚至删除其中的数据和记录，这样会使系统运行出错。针对病毒的各种攻击，病毒防御机制可采取存储映像、数据备份、修改许可、区域保护、动态检疫等方式来保护敏感数据结构。

9.2.6 监控和审计日志

1. 监控

监控可以检测和发现那些可能违反系统安全的活动。例如，在分时系统中，记录一个用户登录时输入的不正确口令的次数，当超过一定的数量时，就表示有人在猜测口令，可能就是非法的用户。这是一种实时的监控活动。

另一种监控活动是周期性的对系统进行全面的扫描。这种扫描一般在系统比较空闲的时间段内进行，这样就不会影响系统的工作效率。可以对系统的以下各个方面进行扫描：

（1）对用户口令进行扫描，找出那些太短的、易于猜测的口令，以提示用户及时改正。

（2）系统目录中是否存在未经授权的程序。

（3）是否存在不是预期的、长时间运行的进程。

（4）用户目录和系统目录是否处于适当的保护状态。

（5）系统的数据文件是否处于一种适当的保护状态。这些文件包括口令文件、设备

驱动程序以及操作系统的内核本身。

（6）是否存在危险的程序搜索路径入口（如特洛伊木马程序）。

由系统安全扫描发现的问题可以由系统自动修复，也可以报告给系统管理员，由管理员来解决。

2. 审计日志

日志文件是安全系统的一个重要组成部分，它记录计算机系统所发生的情况：何时由谁做了一件什么样的事，结果如何等。日志文件可以帮助用户更容易跟踪间发性问题或一些非法侵袭，可以利用它综合各方面的信息，去发现故障的原因、侵入的来源以及系统被破坏的范围。对于那些不可避免的事故，也至少对事故有一个记录。因此，日志文件对于重新建立用户的计算机系统、进行调查研究、提供证据以及获得准确及时的现场服务都是必须的。

但是，日志文件有一个致命的弱点：它通常记录在自身系统上，它们会遭受到修改或删除。有些技术方法可以帮助缓解这种问题，但无法完全消除隐患。有些系统支持将日志文件存到不同的机器上，这样对于日志文件的安全就有了很好的保证。

本章小结

本章主要介绍了操作系统安全性的相关问题。人们对计算机系统的安全性很重视，操作系统的安全性是计算机系统安全性的重要组成部分。操作系统的安全功能就是为数据处理系统从技术和管理上采取安全保护措施，以保护计算机硬件、软件和数据不因偶然的因素或恶意的攻击而遭到破坏，使计算机系统能够安全运行。

操作系统的安全机制有很多，而且针对特别的计算机安全问题有专门的保护机制，如内存保护机制、文件保护机制。为防止非法用户存取系统资源，许多操作系统采取了切实可行的、极为严密的安全措施。目前最常用的用户身份认证机制是口令。存取控制通常通过以单实体或单用户为基础的存取控制矩阵来实现。存取控制矩阵有访问控制表和访问权限表两种形式。加密是一项重要的计算机安全技术，加密就是把可理解的信息转换成为不可理解的信息。病毒是具有破坏性的计算机程序，它的运行离不开系统环境的支持，可以采用多种机制来防御病毒，但都有其局限性，都不是完美无缺的。监控可以检测和发现那些可能的违反系统安全的活动；日志文件可以帮助用户更容易跟踪间发性问题或一些非法侵袭，可以利用它综合各方面的信息，去发现故障的原因、侵入的来源以及系统被破坏的范围。

操作系统的安全性是计算机系统安全性的基础，所以设计操作系统时必须考虑其安全性。

习 题 九

9-1 对计算机系统安全性的威胁有哪些？

9-2 系统安全性的内涵是什么？

9-3 操作系统的安全性功能有哪些？

9-4 可信任计算机系统评价标准将计算机系统的安全程度分为哪几个等级？

9-5 CC 标准中有哪几个安全保证级别？

9-6 什么是访问控制表？什么是访问权限表？

9-7 病毒防御机制有哪些？

9-8 监控和审计日志的作用是什么？

9-9 简述数据加密技术。

9-10 如何保证口令的安全？

参考文献

[1] 汤子瀛，哲凤屏．计算机操作系统．西安：西安电子科技大学出版社，2000
[2] 张尧学，史美林．计算机操作系统教程．北京：清华大学出版社，2001
[3] 庞丽萍．操作系统原理．武汉：华中理工大学出版社，1988
[4] 孟静．操作系统教程．北京：高等教育出版社，2001
[5] 孟庆昌．操作系统．北京：中央广播电视大学出版社，2000
[6] 蒋静．操作系统原理·技术与编程．北京：机械工业出版社，2003
[7] Gary Nutt．操作系统现代观点．北京：机械工业出版社，2003
[8] 王素华．操作系统教程．北京：人民邮电出版社，1996 年
[9] 屠立德，屠祁．操作系统基础（第二版）．北京：清华大学出版社，1995
[10] 吴企渊，梁燕．计算机操作系统．北京：清华大学出版社，2000
[11] 袁建红．实用操作系统．北京：机械工业出版社，2002
[12] 柯敏毅．计算机操作系统教程（第二版）．北京：中国水利水电出版社，2002
[13] 何炎祥．计算机操作系统．北京：清华大学出版社，2004
[14] 李善平，郑扣根．操作系统实验教程．北京：机械工业出版社，1999